(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
16 September 2004 (16.09.2004)

PCT

(10) International Publication Number
WO 2004/079631 A2

(51) International Patent Classification[7]: G06K

(21) International Application Number:
PCT/IB2004/050148

(22) International Filing Date: 25 February 2004 (25.02.2004)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
03100517.6 3 March 2003 (03.03.2003) EP

(71) Applicants *(for all designated States except US)*: KONINKLIJKE PHILIPS ELECTRONICS N.V. [NL/NL]; Groenewoudseweg 1, NL-5621 BA Eindhoven (NL). PAUWS, Steffen, C. [NL/NL]; c/o Prof. Holstlaan 6, NL-5656 AA Eindhoven (NL).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: EGNER, Sebastian [DE/NL]; c/o Prof. Holstlaan 6, NL-5656 AA Eindhoven (NL). KORST, Johannes, H., M. [NL/NL]; c/o Prof. Holstlaan 6, NL-5656 AA Eindhoven (NL). VAN VUUREN, Marcel [NL/NL]; Prof. Holstlaan 6, NL-5656 AA Eindhoven (NL).

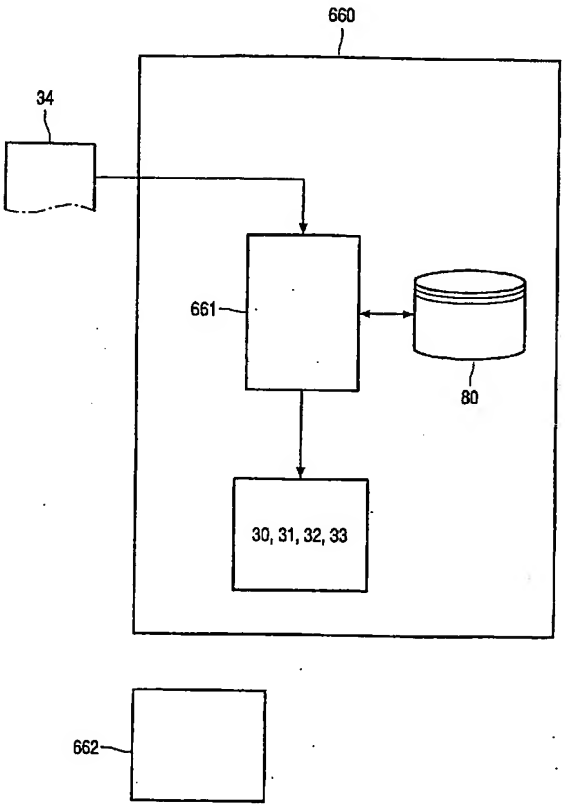
(74) Agent: GROENENDAAL, Antonius, W., M.; Prof. Holstlaan 6, NL-5656 AA Eindhoven (NL).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH,

*[Continued on next page]*

(54) Title: METHOD AND ARRANGEMENT FOR SEARCHING FOR STRINGS




(57) Abstract: This invention relates to methods of searching for a final number of result strings (30-33) having a partial or an exact match with a query string (34) in a database (80) comprised of many long strings or a long string, said method includes the steps of partitioning the query string in a first number of input query strings (35, 36, 37); determining a second number of neighboring strings (38-41, 42-45, 44-49, respectively) for each string in said first number of input query strings, wherein each string in said second number of neighboring strings has a predetermined first number of errors; searching the database for a third number of exact matches (50-61, 70-74) for each string in said second number of neighboring strings based on a search method; concatenating said searched exact matched strings from the database into a fourth number of intermediate strings (29, 30, 32, 33, 34) wherein said searched exact matched strings (50-61, 70-74) comprised in each of said intermediate strings are in succession to one another in said database; and determining the final number of result strings (30-33) based in said fourth number of intermediate strings, wherein each string in the final number of result strings has a maximum of predetermined second number of errors compared to said query string (34). This enables for a perfect match or a partial match containing only minor errors with respect to said query string, and for a fast search in larger databases with a relative low use of processing power.

WO 2004/079631 A2

GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

**Published:**

— *without international search report and to be republished upon receipt of that report*

*For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.*

Method and arrangement for searching for strings

This invention relates to a method of searching for a final number of result strings having a partial or an exact mach with a query string in a database comprised of many long strings or of a long string.

The present invention also relates to a search engine.

The present invention also relates to a tool.

The present invention also relates to a computer system for performing the method.

The present invention further relates to a computer program product for performing the method.

Additionally, the present invention further relates to an arrangement.

US 5,963,957 discloses an information processing system having a music database. Said music database stores homophonic reference sequences of music notes. The reference sequences are all normalized to the same scale degree in order to be stored lexicographically. A so called N-ary is applied to find a match between a string of input music notes and a particular reference sequence. Hereby said information processing system provides bibliographic information associated with the matching reference sequence.

In Du, D.W. and Chang, S.C. (1994), an approach to designing very fast approximate string matching algorithms, IEEE Transactions on Knowledge and Data Engineering, 6, 4, 620-633, another kind of string matching is further disclosed.

In the art, retrieval methods use algorithms for exact matching. However the known retrieval methods typically attempt an exact match, i.e. the search or the match is performed in order to find a perfect match.

However, it is a problem in many practical applications that only a perfect match is searched for. As a result, it is an additional problem that no matching result(s) is/are provided even though this/these may be useful even if it/they only contained minor errors.

It is a further problem that search of large database takes a long time and correspondingly requires intensive usage of processing power.

In many practical applications it is sufficient to obtain a partial match (instead of the perfect match). This is the case since either a query string - as an input to the search

attempt – or the result matching string, both may have less important errors but still it is better to obtain the partial match result than no result at all. Said errors, typically, are caused by improper data comprised either in the query string or in the database searched in.

The above and other problems are solved by said method, when the method comprises the steps of:

- partitioning the query string in a first number of input query strings;

In other words, in this step, the query string is cut into said first number of small pieces of substrings, i.e. into said input query strings.

- determining a second number of neighboring strings for each string in said first number of input query strings, wherein each string in said second number of neighboring strings has a predetermined first number of errors;

In other words, in this step, the second number of neighboring strings depends of the length of the query string, the size of different discrete symbols in the string alphabet applied and the numbers of errors allowed in the neighboring strings.

In general, for each string in said first number of input query strings said second number of neighboring strings are determined. Each of these, individually has a predetermined first number of errors, which is greater than or equal to zero.

- searching the database for a third number of exact matches for each string in said second number of neighboring strings based on a search method;

Hereby, is the database searched for a number (third) of exact matches for each string in said second number of neighboring strings based on a given search method, the search method can be a q-gram index method, a suffix tree method or a hash method.

- concatenating said searched exact matched strings from the database into a fourth number of intermediate strings wherein said searched exact matched strings comprised in each of said intermediate strings are in succession to one another in said database; and

- determining the final number of result strings based on said fourth number of intermediate strings, wherein each string in the final number of result strings has a maximum of a predetermined second number of errors compared to said query string.

For the last two steps, these are explained in figure 5, i.e. step 400 and 500.

As a result of the method, said final number of result strings, each is an exact or a partial match to said query string (mentioned in the opening paragraph).

It is hereby achieved to obtain a perfect match or a partial match containing only minor errors.

Further, the method can search large databases (for perfect or partial matches) fast with a relative low use-of processing power.

Said arrangement, tool, search engine, computer system, respectively provides the same advantages and solves the same problem(s) for the same reasons as described previously in relation to the method.

The invention will be explained more fully below in connection with preferred embodiments and with reference to the drawings, in which:

Fig. 1 shows a general discussion of the art;

Fig. 2 shows a way of partitioning the query string;

Fig. 3 shows a practical detailed example way of partitioning the query string and a subsequent search;

Fig. 4 shows a general example way of partitioning and searching the query string;

Fig. 5 shows a method of searching for a final number of result strings, and

Fig. 6 shows an arrangement for searching.

Throughout the drawings, the same reference numerals indicate similar or corresponding features, functions, strings, etc.

Figure 1 shows a general discussion of the art. The figure shows a query string 'abacababc' searched for in a string database, reference numeral 80. The database positions of the four approximate matches of the query string allowing at most one error ($k = 1$) are indicated. The query string, reference numeral 34, 'abacababc' which is composed from a 3-letter alphabet {'a','b','c'}. If only one error ($k = 1$) is allowed, an approximate match is found containing an added symbol (e.g., reference numeral 30: 'abacadabbc'), an approximate match containing a deleted symbol (e.g., reference numeral 31: 'abcababc'), an approximate match containing a substituted symbol (e.g., reference numeral 32: 'abacadcbc') and an exact match (i.e., reference numeral 33: 'abacadabc'). It is generally acknowledged in the art to search for a string whereby the entire query string is searched for at once.

Figure 2 shows a way of partitioning the query string. Efficient retrieval methods, known from the literature, capitalize on the use of fast algorithms for exact matching, that is, searching without allowing any error (or, $k = 0$). This would only return the position for (exact) match, i.e. reference numeral 33 from figure 1. Specialized index structures such as suffix trees or $q$-grams need to be build off-line (as a pre-processor) from the string database, reference numeral 80 to enable the implementation of fast exact matching algorithms. Essentially, these structures keep hold of all positions of each small sub-string

that occur in the database. Effectively, this means that the retrieval process can immediately jump to the relevant spots in the database while discarding the irrelevant parts. The $q$-gram index method is used, since it is more space-efficient (i.e., uses less memory) than other methods and can be easily adapted to our purpose. The $q$-gram keeps all positions of all sub-strings of length $q > 0$ occurring in the database. If, for instance, the database consists of the string 'abababcabcabacab.....', the q-gram with $q = 4$ collects the starting positions of all sub-strings of length 4 such as 'abab', 'baba', 'abab', 'babc' and 'abca'. These sub-strings are indexed by using a function, listed and sorted in an easily accessible data structure. Similar sub-strings, such as 'abab' in the example, end up at the same index (called a bucket). A $q$-gram allows us to obtain the database positions of all exact matches of a query of length $m \leq q$ by computing the index function and retrieve the members of a bucket. Queries that are longer than $q$ need an additional check because only the prefixes of length $q$ of the query can be in a bucket. The standard way to work with $q$-grams is described by Myers, 1994.

As alternatives to said q-gram method, a suffix tree method or a hash method may be applied.

If exact matching methods are used in approximate matching, a workaround has to be found for the errors allowed. For instance, according to the invention a set of strings can be generated that differ on a limited number of locations in the original query (string). These strings are called neighbors of the query. Neighbors thus represent the errors in a query. Formally, a $k$-neighborhood of a string $S$ is defined as the set of strings with at most $k$ errors with respect to $S$. For instance, if the query 'abba' is constructed from the two-letter alphabet {'a','b'}, the complete set of neighbors with at most one error (i.e., error level $k \leq 1$) consists of the original 'abba', all strings with a deletion 'abb', 'aba' and 'bba', all strings with a addition 'aabba', 'babba', 'abbba', 'ababa', 'abbaa' and 'abbab' and all strings with a substitution 'bbba', 'aaba', 'abaa' and 'abbb'.

Neighbors of a given string can be efficiently generated (Myers, 1994). If the neighbors are located in the database using the $q$-gram method, these exact matches correspond to approximate matches of the original query.

However, the number of neighbors to be investigated is exponential when longer queries are investigated, a larger alphabet and a higher error level. To resolve this issue and gain retrieval speed, the query is first partitioned in smaller sub-strings and for each sub-string its set of neighbors is generated. Then all these neighbors are searched exactly using $q$-grams or the other search methods mentioned. Their exact matches in the database correspond now to *partial* approximate matches of the original query.

Let the query be 'abacababc' composed from the three-letter alphabet {'a','b','c'} and allow an error level of 3 ($k = 3$). Note that errors in the query can occur at any place. For instance, the errors can be

- all at the start (e.g., 'ccccababc' can then be the string you are actually looking for),
- all in the middle (e.g. 'ababbcabc'),
- all at the end (e.g., 'abacabbca')
- or uniformly distributed in the query (e.g., 'abccacabb').

If the query is partitioned into three parts ($p = 3$, which results in the substrings 'aba', 'cab' and 'abc' for our example query 'abacababc'), a set of neighbors is generated for each part and each neighbor is searched *individually* in the database, context is lost about how the original query looks like. To understand this, note that the neighbors can occur anywhere in the database. Neighbors do not necessarily appear close together or in a sequence, which is required to form an approximate match of the original query. In other words, when an exact match is found for a neighbor for our example query 'abacababc', it cannot be known whether it reflect the first, second or third part of the query and what kind of neighbors are found for the other parts. Necessary measures have to be taken to reveal this information. Former methods described in literature stop right here, that is, they consider each exact match of a neighbor as a useful candidate for resolving the query. In contrast, the invention of *cross-cutting* does an additional filter step by re-establishing the (error) context while searching the neighbors. The neighbors that can be discarded are

- those that do not appear in a sequence with other neighbors in the database, and
- those that form a sequence with other neighbors in the database that *cannot* be an approximate match of the original query.

These observations are condensed in a central 'cross-cutting' lemma for the current invention that provides a guarantee about the successful retrieval of relevant parts if we partition a query string in $p$ parts and search for the parts separately each with at most $k_i$ errors.

Cross-cutting lemma: Let $A$ and $B$ be two strings such that the number of errors between them is smaller than or equal to $k$ in edit-distance sense, or formally $\partial(A,B) \leq k$ . Let $A = A_1 A_2 \cdots A_p$ be the partitioning $A$ in $p$ parts, for strings $A_i$ and for any $p > 1$. Let $K = (k_1, k_2, \cdots, k_p)$ be any sequence of nonnegative numbers such that

$C = \sum_{i=1}^{p} k_i + p - k \geq 1$. Then there exists a partitioning $B = B_1 B_2 \cdots B_p$ and a subset of the parts indexed by $J = (j_1, j_2, \cdots, j_l)$ such that $\sum_{i=1}^{l} k_{j_i} + 1 - \partial(A_{j_i}, B_{j_i})) \geq C$.

Proof: It is evident that $B$ can be partitioned in $p$ parts such that $\sum_{i=1}^{p} \partial(A_i, B_i) = \partial(A, B)$: look to it that the errors are located at corresponding parts and do not introduce additional errors. We choose such a partitioning of $B$ and all parts $i$ in the subset $J$ for which $k_i + 1 \geq \partial(A_i, B_i)$. Now we can derive the following which proves our case:

$$C = \sum_{i=1}^{p} k_i + p - k \leq \sum_{i=1}^{p} k_i + 1 - \partial(A_i, B_i) \leq \sum_{i=1}^{l} k_{j_i} + 1 - \partial(A_{j_i}, B_{j_i})$$

The lemma is used as a filtration condition in the invention with A being the query string and B the database string. The lemma says that the number of errors $e$, that a sequence of neighbors in the database represents, has to fulfil a particular criterion. For that criterion, $k_i$ is applied as the pre-defined number of errors that are allowed in each sub-string $i$ of the query. Then, the error summation $\sum_{i=1}^{p} (k_i + 1) - e$ should be at least a constant $C = \sum_{i=1}^{p} k_i + p - k$ to be still part of an approximate match of the query. In these formulae, $p$ is the number of sub-strings in which the query string is partitioned, $k_i$ is the number of errors allowed in each sub-string $i$, and $k$ is the maximum total number of errors (error level).

Calculating the error summation $\sum_{i=1}^{p} (k_i + 1) - e$ and comparing it to $C$ is the foundation of the cross-cutting algorithm. Shortly, each time a new match of a neighbor has been found at a particular position in the database, it is checked whether or not there are matches of other neighbors preceding that database position. Specialized data structure keeps track of all the positions of the neighbors in the database in an efficient way. Then, it is verified whether or not the concatenation of these consecutive matches (as they appear in the database) can eventually result in an approximate match of the complete query. If the error summation is equal or larger than the threshold C, than these neighbors are still relevant candidates for being part of an approximate match of the query. If the error summation is lower than this threshold C, all involved neighbors can be discarded.

As shown in figure 2, the query 'abacababc', reference numeral 34, is partitioned in three sub-strings ($p=3$). Recall that only 3 errors are allowed ($k=3$). Define $k_i=\left\lfloor\frac{k}{p}\right\rfloor=1$ and $C=\sum_{i=1}^{p}k_i+p-k=3$. For each sub-string, a set of neighbors i.e. reference numeral 38-41, 42-45 and 46-49, respectively is generated which are sought exactly in the database. In the process of neighbor search, the positions of all neighbors found so far are kept and it is decided whether or not the concatenation of consecutive neighbors can be part of an approximate match of the query. The two matches for the neighbors 'aba' and 'cab' (see reference numeral 30 in figure 2) represent an error-free match of the first two sub-strings of the query string (i.e., $e=0$ and $\sum_{i=1}^{2}(k_i+1)-e=4\geq C=3$). Already it is known that a valid approximate match of the query have been found; in the worst case a neighbor representing 3 errors can succeed these two matches for which our filtration condition still holds. The two matches for the neighbors 'abc' and 'caa' (see 31 in figure 2) represent 2 errors (i.e., $e=2$ and $\sum_{i=1}^{2}(k_i+1)-e=2$). This sequence can only be succeeded by a neighbor representing at most 1 error to be still a valid approximate match of the query. The matches for the neighbors 'acb' and 'cbc' (see reference numeral 32 in figure 2) represent already 4 errors (i.e., e = 4 and) $\sum_{i=1}^{2}(k_i+1)-e=0$. It is already known that this sequence cannot be part of an approximate match of the query. Even if it is succeeded by an error-free neighbor, the filtration condition does not hold.

For a more detailed and general discussion of said Q-grams and said Neighboring generation, respectively; the followings sections should enable the person skilled in the art to implement the invention:

**Q-grams or the q-gram index method:**

With q-grams it is possible to find all occurrences of a string not bigger than $q$ very fast. Those q-grams are constructed in the following way.

Consider a bijection $\phi$ of the symbols in $\Sigma$ to the integers 0 to $\sigma-1$.

Function $\phi$ is naturally extended to strings with the recursive definition $\phi(Pa)=\sigma\phi(P)+\phi(a)$ where $P$ is a string over $\Sigma$ and $a$ is a symbol in $\Sigma$. For

$b \in [0, \sigma^q - 1]$, let $Bucket(b) = \{i : \phi(a_i a_{i+1} \cdots a_{i+q-1}) = b\}$. That is, $Bucket(b)$ gives the indices of the leftmost character of each occurrence in $A$ of the unique $q$-symbol string whose $\phi$-code is $b$.

The index is produced in the following way. First $\phi_i = \phi(a_i a_{i+1} \cdots a_{i+m-1})$ is computed for every index $i$. This is done in an $O(n)$ sweep of $A$ using the observation that $\phi_i = a_i \sigma^{m-1} + \lfloor \phi_{i+1} / \sigma \rfloor$. With an $O(n \log(n))$-quick-sort the list can now be produced $Indices = \langle i_1, i_2, \cdots, i_n \rangle$ such that $\phi_{i_j} \leq \phi_{i_{j+1}}$. Finally, the array $Header[b] = \min\{j : \phi_{Indices[j]} = b\}$ is produced in an $O(n)$ sweep of *Indices*. The arrays *Indices* and *Header* provide a realization of the *Bucket* sets. Namely, $Bucket[b] = \{Indices[j] : j \in [Header[b], Header[b+1] - 1]\}$.

If a query $P$ is of length $m \leq q$ then all indices of the occurrences of $P$ are exactly the content of $Bucket(b)$ for $b \in [\phi(P)\sigma^{q-m}, (\phi(P)+1)\sigma^{q-m} - 1]$.

If a query $P$ is larger than q it is known that the occurrences of $P$ must be a subset of those in $Bucket(\phi(P_q))$ where $P_q$ denotes the string consisting of the first $q$ symbols of P.

**Neighbor generation:**

A (complete) $k$-neighborhood of a string $P$ is defined as the set of all strings with a (edit-)distance less than or equal to $k$ from P, i.e. $N_k(P) = \{Q : \partial(Q, P) \leq k\}$.

A condensed $k$-neighborhood of a string $P$ is defined as the set of all strings in the complete $k$-neighborhood of $P$ that do not have a prefix in that neighborhood, i.e., $C_k(P) = \{Q : Q \in N_k(P) \text{ and no prefix of } Q \text{ is in } N_k(P)\}$.

Myers' algorithm computes a condensed k-neighborhood of a string efficiently by generating words from an alphabet and computing the corresponding rows of the dynamic programming matrix of the currently generated word and P. If a word whose last entry of the current row equals k, then a word in the condensed k-neighborhood has been reached. If all entries are larger than k, the algorithm can backtrack. The use of failure links prevents the algorithm from missing words that are in the k-neighborhood but are not in the condensed one.

As the present invention needs the complete k-neighborhood of a string to find all exact matches of the partitions in the database, Myers's algorithm has been adapted.

Figure 3 shows a practical detailed example way of partitioning the query string and a subsequent search.

The query string, reference numeral 34, again is searched for in the database, reference numeral 80. According to the invention, said query string is partitioned into a number of input query strings, the number three is here chosen for conciseness, it may be any other number greater than one. In the example, said input query strings are represented by reference numeral 35, 36 and 37, respectively for the beginning part, the middle part and the "end part" - part.

By means of said number of input query strings, a number of neighboring strings - here four – is defined for each input query string. I.e. for the input query string of reference numeral 35, corresponding four neighboring strings, reference numeral 38, 39, 40 and 41 are determined.

Correspondingly, for the "mid part" input query string of reference numeral 36, corresponding four neighboring strings, reference numeral 42, 43, 44 and 45 are determined.

Correspondingly, for the "end part" input query string of reference numeral 37, corresponding four neighboring strings, reference numeral 46, 47, 48 and 49 are determined.

To the right of the dotted line, reference numeral 80, it is implied that in this section of the figure, the database - previously also indicated by the same reference numeral - is searched into, i.e. said neighboring strings, reference numeral 38 – 49, each is searched for in order to find exact (sub string) match(es).

These are indicated by following the arrows further to the right; as an example, reference numeral 38, a first part neighboring string gives the exact match of reference numeral 50; as another example, reference numeral 47, an end part neighboring string gives the match of reference numerals 58 and 61, and reference numeral 45, a mid part neighboring string gives the "none- useful" result of reference numeral 72.

And in order to achieve final search result(s), more or less matching the query string, reference numeral 34; the arrows can be followed further to the right, i.e. reference numeral 30-33, respectively indicate each of the four final search results. As can be seen, each of said final search results is always comprised of one of the searched beginning substrings, reference numeral 50-53, one of the mid searched substrings, reference numeral 54-57 and one of the "end part" searched substrings, reference numeral 58-61. How these are put in succession and by which criterions will be explained by means of figure 5.

Figure 4 shows a general example way of partitioning and searching the query string. Figure 4 corresponds to figure 3, however, generally the ".." 's indicate that any string of any reference numeral may comprise fewer or more letters, i.e. the invention can be applied for very short strings and for very long sequence of letters as well.

The letters of the western alphabet – as shown - may alternatively be a sequence of elements in a pitch alphabet, a sequence of elements in a musical pitch interval alphabet, a sequence of elements in a musical time interval alphabet, a sequence of binary digits, words or bytes, a sequence of amino acids or a sequence of DNA/RNA. Correspondingly, the same applies for the database searched in, since it may also be understood as one long string or of many long strings.

Said sequence of elements in a musical pitch interval alphabet and sequence of elements in a musical time interval alphabet represents the essence of a musical score. In general for all strings (the query string, strings in the database, etc) this means that these can be made out of any alphabet of discrete symbols.

Figure 5 shows a method of searching for a final number of result strings. The method searches for a final number of result strings as indicated by means of reference numerals 30 through 33 (in the foregoing figures). I.e. each of said final number of result strings will have a partial or an exact mach with the query string, reference numeral 34 in the database, if possible. The database, reference numeral 80, is comprised of a long string. Said method comprises the following steps:

In step 100, the query string is partitioned in a first number of input query strings. As indicated in the forgoing figures, said query string is partitioned in three input query strings, reference numerals 35, 36 and 37, i.e. the first number is here three. Said first number may be any number greater than or equal to one. The first number equaling exact three is only shown for illustrative purposes, i.e. any higher or lower number may be chosen as well.

In other words, in this step, the query string is cut into (said first number) small pieces of substrings, i.e. into said input query strings.

In the example, the query string, reference numeral 34, "aba .. cab.. abc" is cut into said set of first number of input query strings, in the example there are three in the set, i.e. input query string 1, reference numeral 35 = "aba..", input query string 2, reference numeral 36 = "cab.." and input query string 3, reference numeral 37 = "abc..".

In step 200, a second number of neighboring strings are determined. As also indicated in the forgoing figures, said second number of neighboring strings is four, i.e.

reference numerals 38-41 for the first input query string reference numeral 35, reference numerals 42-45 for the second or mid input query string, reference numeral 36 and reference numerals 44-49 for the third or last input query string, reference numeral 37.

The second number equaling exact four is only shown for illustrative purposes, i.e. any higher or lower number than four may be chosen as well. In particular, the number of neighboring strings depends of the length of the query string, the size of different discrete symbols in the string alphabet applied and the numbers of errors allowed in the neighboring strings.

This gives, as the example a total of twelve neighboring strings, i.e. it equals said first number times said second number, i.e. 3 x 4 = 12, i.e. four for each (three) input query strings; or in general: for each string in said first number of input query strings said second number of neighboring strings are determined. As indicated in the forgoing figures, these are the reference numerals 38 through 49. Each of these, individually has a predetermined first number of errors, which is greater than or equal to zero.

Note that If the (first) number of errors exceeds the neighboring string length (i.e. all content of the string is then determined to be in error), consequently a subsequent search in the next step would be absolutely meaning less; therefore said first number of errors cannot exceed the string length.

As the example given, on basis of the input query string "aba..", i.e. reference numeral 35, four neighboring strings are determined, i.e.:

- reference numeral 38 "aba.." equaling itself, .i.e. of course no errors.
- reference numeral 39 "abc.." with one error,
- reference numeral 40 "abb.." with another error, and
- reference numeral 41 "acb.." with two errors.

In the example given, the predetermined first number of errors (which is greater than or equal to zero) is here 0, 1 or 2.

The first number of predetermined of errors, in the example equaling zero, one or two, is only shown for illustrative purposes, i.e. any higher number may be chosen as well.

In step 300, the database is searched for a third number of exact matches for each string in said second number of neighboring strings. Said search is based on a given search method.

Said third number of exact matches is illustrated by means of reference numerals 50-61 and 70-74. It is important to note that there may be one or more matches, e.g.:

- firstly, reference numeral 38 "aba.." - as an example of a neighboring string - leads to the exact match of reference numeral 50 and 52, i.e. "aba..";
- secondly, reference numeral 39 "abc.." - as another example of a neighboring string - leads also to its exact match of reference numeral 51 "abc..",
- thirdly, reference numeral 40 "abb" .." leads also to no match, i.e. reference numeral 70 "abd..", and
- finally, reference numeral 41 "acb" .." also leads to no match, i.e. reference numeral 71 "abc..".

The latter two is then unusable since only exact matches are considered.

Correspondingly, reference numerals 53-61 and 72-74 are search results from the neighboring strings indicated by means of reference numerals 42 – 49.

In all cases, the search results, reference numerals 50-61 and 70-74, with corresponding string content and corresponding positions in the database are kept for later use in the subsequent step.

Also, in all cases, the given search method may be the q-gram index method or any other appropriate method known to be useful in the art, e.g. a suffix tree method or a hash method.

In step 400, said searched exact matched strings from the database are concatenated into a fourth number of intermediate strings. Said searched exact matched strings (when comprised in each of said intermediate strings) are in succession to one another in said database.

Said fourth number of intermediate strings is indicated by means of reference numerals 29 - 33, the fourth number in the example given is five. Further, said searched exact matched strings - indicated by means of reference numerals 50-61 and 70-74 – comprised in each of said intermediate strings are determined to be in succession to one another in said database: this will be explained in the following:

As can be seen from the examples – during concatenation – search result from the first input query string (being the beginning of the query string), reference numeral 35 = "aba.." having corresponding beginning neighboring strings, reference numeral 38 – 41 lead to corresponding beginning substrings, reference numeral 50-53.

Correspondingly, – during concatenation – search result from the second input query string (being the middle of the query string), reference numeral 36 = "cab.." having corresponding "middle" neighboring strings, reference numeral 42 – 45 lead to corresponding mid substrings, reference numeral 54-57.

Correspondingly, – during concatenation – search result from the third input query string (being the end part of the query string), reference numeral 37 = "abc.." having corresponding "end part" neighboring strings, reference numeral 46 – 49 lead to corresponding "end part" substrings, reference numeral 58-61.

In other words, the exact matched strings, reference numerals 50-61 and 70-74 as corresponding part of each of said intermediate strings are in fact in succession to one another in the database, i.e. one of the beginning substrings, reference numeral 50-53, one of the mid substrings, reference numeral 54-57 and one of the end part substrings, reference numeral 58-61 are concatenated into one of said fourth number of intermediate strings, i.e. one of reference numerals 29-33.

In step 500, the final number of result strings is determined. The determination is based on said fourth number of intermediate strings from the foregoing step, and here – in this step - each string in the final number of result strings is determined to have a maximum of a predetermined second number of errors compared to said query string of reference numeral 34.

In the examples given, the final number of result strings, reference numerals 30-33, is four, whereas said fourth number of intermediate strings was five. I.e. in the example, one, reference numeral 29 is discarded or left out since this in particular does not satisfy the criterion of having error(s) less than or equal to said second number of errors. This is seen when compared to said query string, of reference numeral 34.

In other words, reference numeral 29 is discarded for having too many errors (compared the initial query string of reference numeral 34), whereas reference numerals 30 through 33, individually had less errors and thus satisfied the criterion. That is, in the example reference numerals 30 through 33 comprise the final number of result strings.

As a result of the method, said final number of result strings, reference numerals 30 - 33, each is an exact or a partial match to said query string, reference numeral 34.

In the example given, four matches (exact or partial) was then the result when said query string was searched for.

This step - in connection with the foregoing step – is also called "Cross-cutting", i.e. the idea of considering only those exact matches of neighbors (when searched for) that, when concatenated, can contain an approximate match with the original query string of reference numeral 34.

In the spirit of the invention, any of said "first number", "second number of neighboring strings", "third number", "fourth number of intermediate strings" and "first and second number of errors, may be fine tuned individually or in relation to one another or in relation to the content of the query string and / or the database. Hereby either speed of search may be decreased and / or another matching (fewer/less) errors may be obtained.

Correspondingly, the given examples are illustrative and may also be expanded with another string length for the query string, neighboring strings, intermediate strings, another sequential content (of discrete symbols) of the strings, etc.

Figure 6 shows an arrangement for searching. Reference numeral 660 indicates said arrangement. The arrangement processes - according to the invention – the query string, reference numeral 34 as discussed in the foregoing figure. The arrangement processes said string as input and therefore comprises calculation means 661, e.g. a sufficient fast microprocessor. The microprocessor searches for matches in the database, reference numeral 80. As a result, if any, the final number of result strings, reference numeral 30, 31, 32 and 33 is found. The calculation means for carrying out the steps of the matching method can also be e.g. a part of a dedicated ASIC.

Reference numeral 662 denotes a computer program product. Said computer program product comprising program code means stored on a computer readable medium for performing said method when the computer program is run on a computer.

Generally, the present invention may be applied in various fields, such as melody retrieval for music systems ('query by humming'), finding keywords in a search engine or in a text file, finding DNA/RNA sequences in a molecular biology database, bits, bytes or word codes, error control, etc. For the melody retrieval application, it can be imagined that only a small fragment of a melody is remembered without recalling the complete melody or song. Once provided in the appropriate representation as a string of discrete symbols, this melody fragment can then be input to the search method to reveal the identity of the song or melody using said database. The search method thereby allows for errors in the input. The arrangement may be e.g. a jukebox –implemented as a stand alone audio apparatus or on a PC. It may also be a portable audio apparatus, which contains an interface enabling e.g. a jogger to quickly change his accompanying music by whistling the beginning or refrain. The arrangement may e.g. also be a service on an internet server for quickly selecting a particular MP3 from the web, or the arrangement may be a portable phone running the method, for retrieving e.g. a ring tone.

Similarly, keywords as a query (similar to previously mentioned query string) for a search engine –e.g. for searching a particular product on Internet or a word in a software dictionary- or a software search tool can contain typos. The search or retrieval process can account for these errors in the keywords. In all applications, the number of errors one allows may be pre-defined and fixed. The database can be best seen as one very long string (e.g., a long text, all melodies in the world put into sequence, etc). Also, the strings may be constructed from a finite collection (e.g. western or pitch alphabet, binary digits, bytes, words, amino acids, DNA/RNA, words, etc). For a text application, the 26 letters from the Western alphabet may be used. Correspondingly, melodies can be constructed from a 9-element pitch interval alphabet. Molecular biology applications use the twenty amino acids or the four nucleotides as the alphabet. Coding applications use the binary symbols, word, bits or bytes.

Under the essence of a musical score should be understood any information which is sufficient for retrieving a melody. E.g. tone intervals, or just time intervals in case a person is not very musical or wants to search a piece of music by e.g. tapping his foot, or both. These are converted into string characters by a predetermined mapping function.

A computer readable medium may be magnetic tape, optical disc, digital versatile disk (DVD), compact disc (CD record-able or CD write-able), mini-disc, hard disk, floppy disk, smart card, PCMCIA card, etc.

In the claims, any reference signs placed between parentheses shall not be constructed as limiting the claim. The word "comprising" does not exclude the presence of elements or steps other than those listed in a claim. The word "a" or "an" preceding an element does not exclude the presence of a plurality of such elements.

The invention can be implemented by means of hardware comprising several distinct elements, and by means of a suitably programmed computer. In the device claim enumerating several means, several of these means can be embodied by one and the same item of hardware. The mere fact that certain measures are recited in mutually different dependent claims does not indicate that a combination of these measures cannot be used to advantage.

Reference:

Myers, E. (1994). A sublinear algorithm for approximate keyword searching. Algorithmica, 12 (4/5), 345-374.

CLAIMS:

1. A method of searching for a final number of result strings (30-33), having a partial or an exact match with a query string (34), in a database (80) comprised of many long strings or a long string, said method comprising the steps of:

- partitioning (100) the query string in a first number of input query strings (35, 36, 37);
- determining (200) a second number of neighboring strings (38-41, 42-45, 44-49, respectively) for each string in said first number of input query strings, wherein each string in said second number of neighboring strings has a predetermined first number of errors;
- searching (300) the database for a third number of exact matches (50-61, 70-74) for each string in said second number of neighboring strings based on a search method;
- concatenating (400) said searched exact matched strings from the database into a fourth number of intermediate strings (29, 30, 32, 33, 34) wherein said searched exact matched strings (50-61, 70-74) comprised in each of said intermediate strings are in succession to one another in said database; and
- determining (500) the final number of result strings (30-33) based on said fourth number of intermediate strings, wherein each string in the final number of result strings has a maximum of a predetermined second number of errors compared to said query string (34).

2. A method according to claim 1 characterized in that said search method is a q-gram index method.

3. A method according to claim 1 characterized in that said search method is a suffix tree method.

4. A method according to claim 1 characterized in that said search method is a hash method.

5. A method according to any one of claims 1 to 4 characterized in that said strings and said database each comprises a sequence of letters of the western alphabet.

6. A method according to any one of claims 1 to 4 characterized in that said strings and said database each represents the essence of a musical score.

7. A method according to any one of claims 1 to 4 characterized in that said strings and said database each comprises a sequence of binary digits.

8. A method according to any one of claims 1 to 4 characterized in that said strings and said database each comprises a sequence of amino acids or a sequence of DNA/RNA bases.

9. A method according to any one of claims 1 to 4 characterized in that said strings and said database each comprises a sequence of bits, bytes or words.

10. A search engine comprising calculation means (661) for carrying out the steps of the methods according to clams 1 to 9.

11. A tool comprising means for carrying out the steps of the methods according to claims 1 to 9.

12. An arrangement (660) comprising:

- calculation means for (661) partitioning the query string in a first number of input query strings (35, 36, 37);
- calculation means for (661) determining a second number of neighboring strings (38-41, 42-45, 44-49, respectively) for each string in said first number of input query strings, wherein each string in said second number of neighboring strings has a predetermined first number of errors;
- calculation means for (661) searching the database for a third number of exact matches (50-61, 70-74) for each string in said second number of neighboring strings based on a search method;
- calculation means for (661) concatenating said searched exact matched strings from the database into a fourth number of intermediate strings (29, 30, 32, 33, 34) wherein

said searched exact matched strings (50-61, 70-74) comprised in each of said intermediate strings are in succession to one another in said database; and

- calculation means for (661) determining the final number of result strings (30-33) based on said fourth number of intermediate strings, wherein each string in the final number of result strings has a maximum of a predetermined second number of errors compared to said query string (34).

13. A computer system for performing the method according to any one of claims 1 through 9.

14. A computer program product (662) comprising program code means stored on a computer readable medium for performing the method of any one of claims 1 through 9 when the computer program is run on a computer.

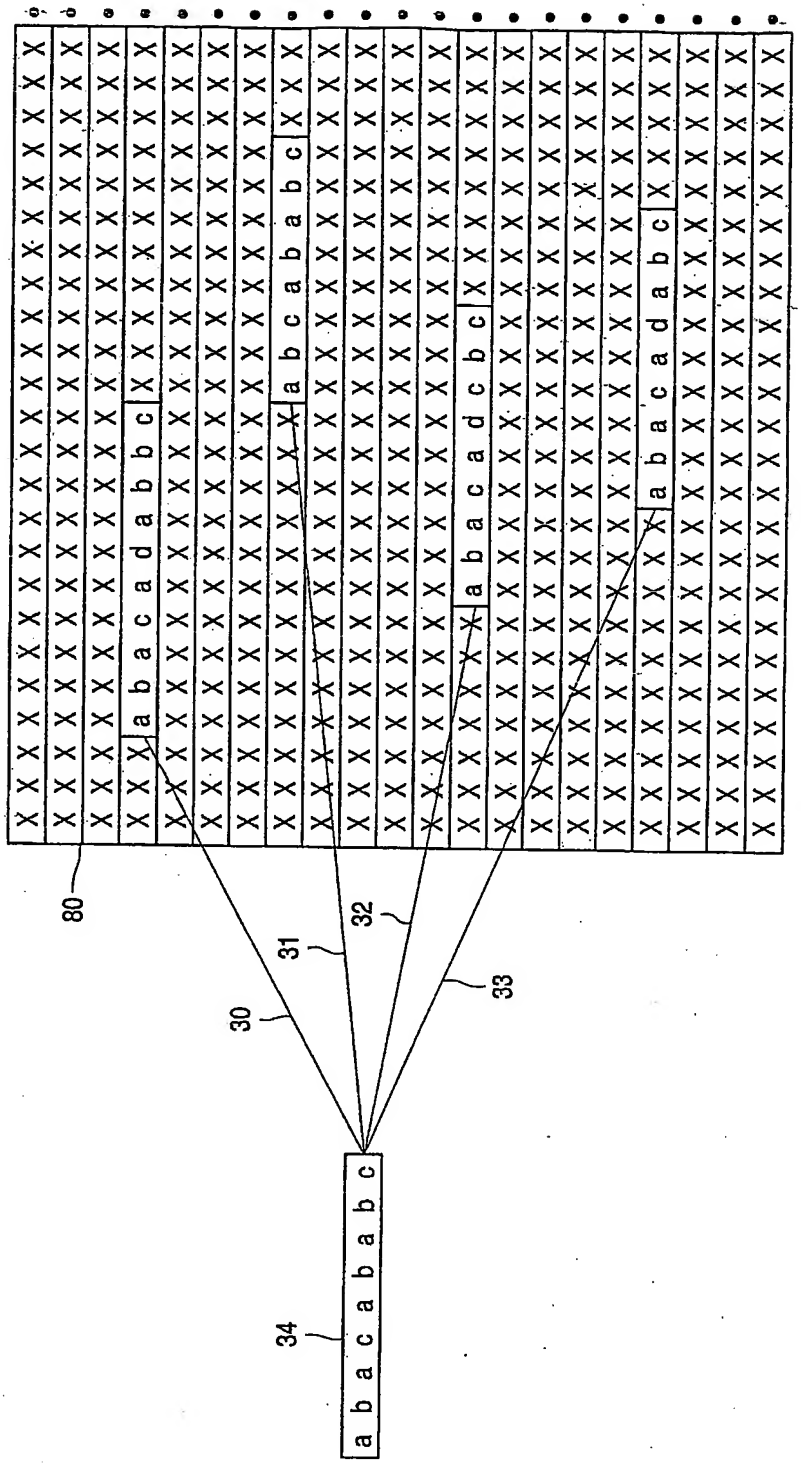

a b a c a d a b b c
a b c a b a b c
a b a c a d c b c
a b a c a d a b c
a b a c a b a b c
80
31
32
30
33
34

FIG. 1

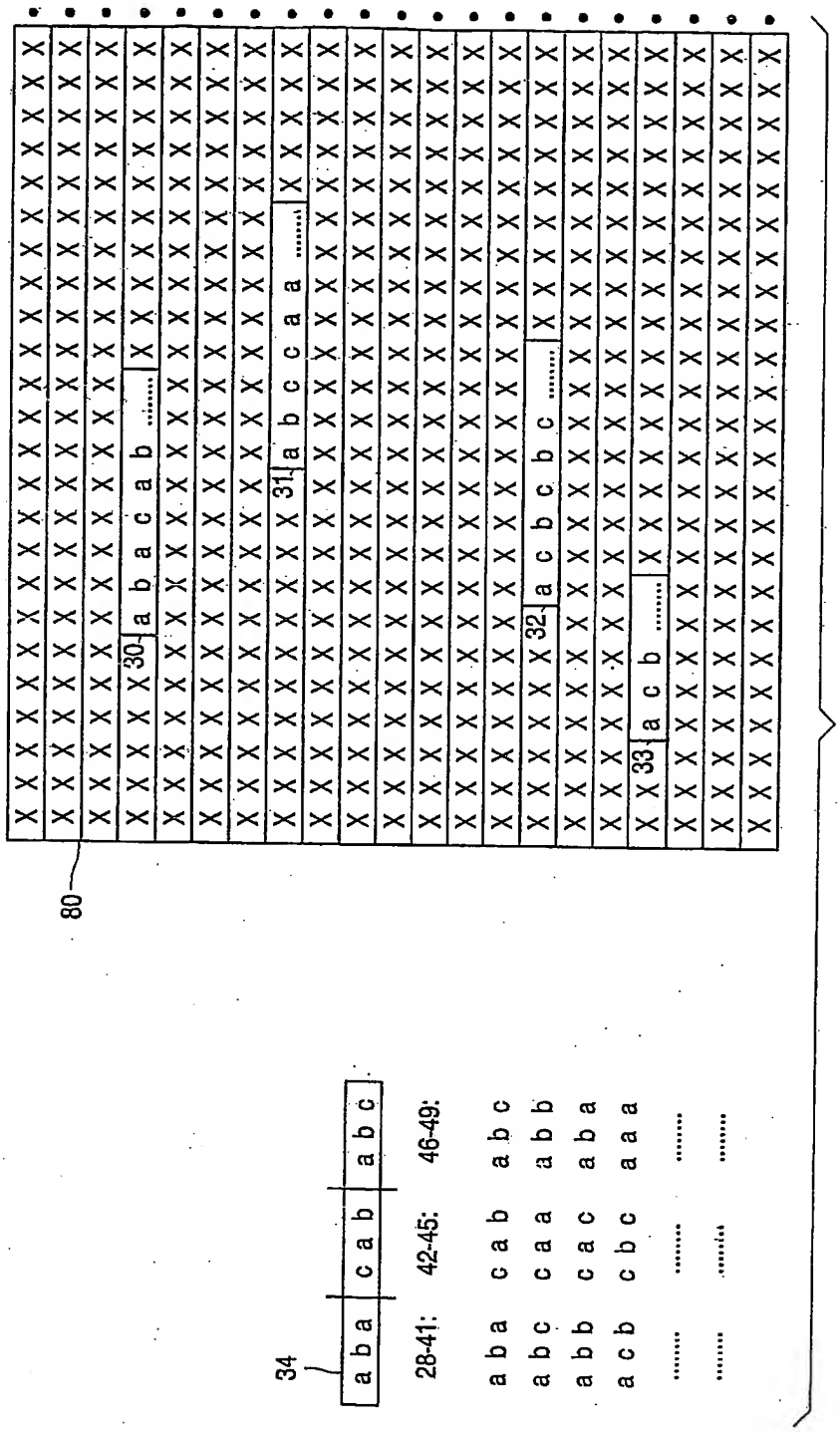

80
34
aba cab abc
28-41: 42-45: 46-49:
aba cab abc
abc caa abb
abb cac aba
acb cbc aaa

FIG. 2

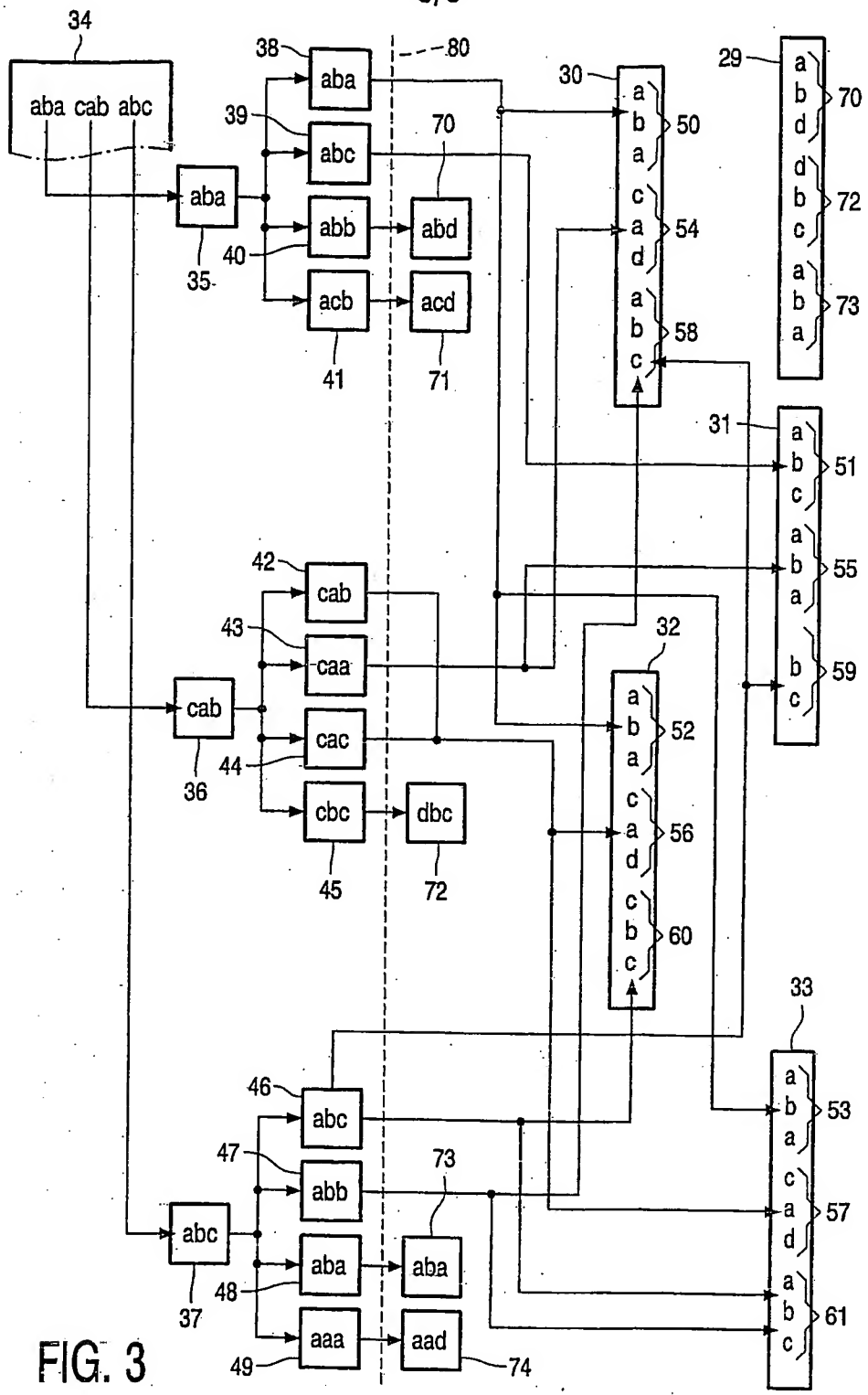

34
aba cab abc
38
aba
39
abc
70
aba
35
abb
abd
40
acb
acd
41
71
80
30
a
b
a
50
c
a
d
54
a
b
c
58
29
a
b
d
70
d
b
c
72
a
b
a
73
31
a
b
c
51
a
b
a
55
b
c
59
42
cab
43
caa
cab
36
cac
44
cbc
dbc
45
72
32
a
b
a
52
c
a
d
56
c
b
c
60
33
a
b
a
53
c
a
d
57
a
b
c
61
46
abc
47
abb
73
abc
37
aba
aba
48
aaa
aad
49
74

FIG. 3

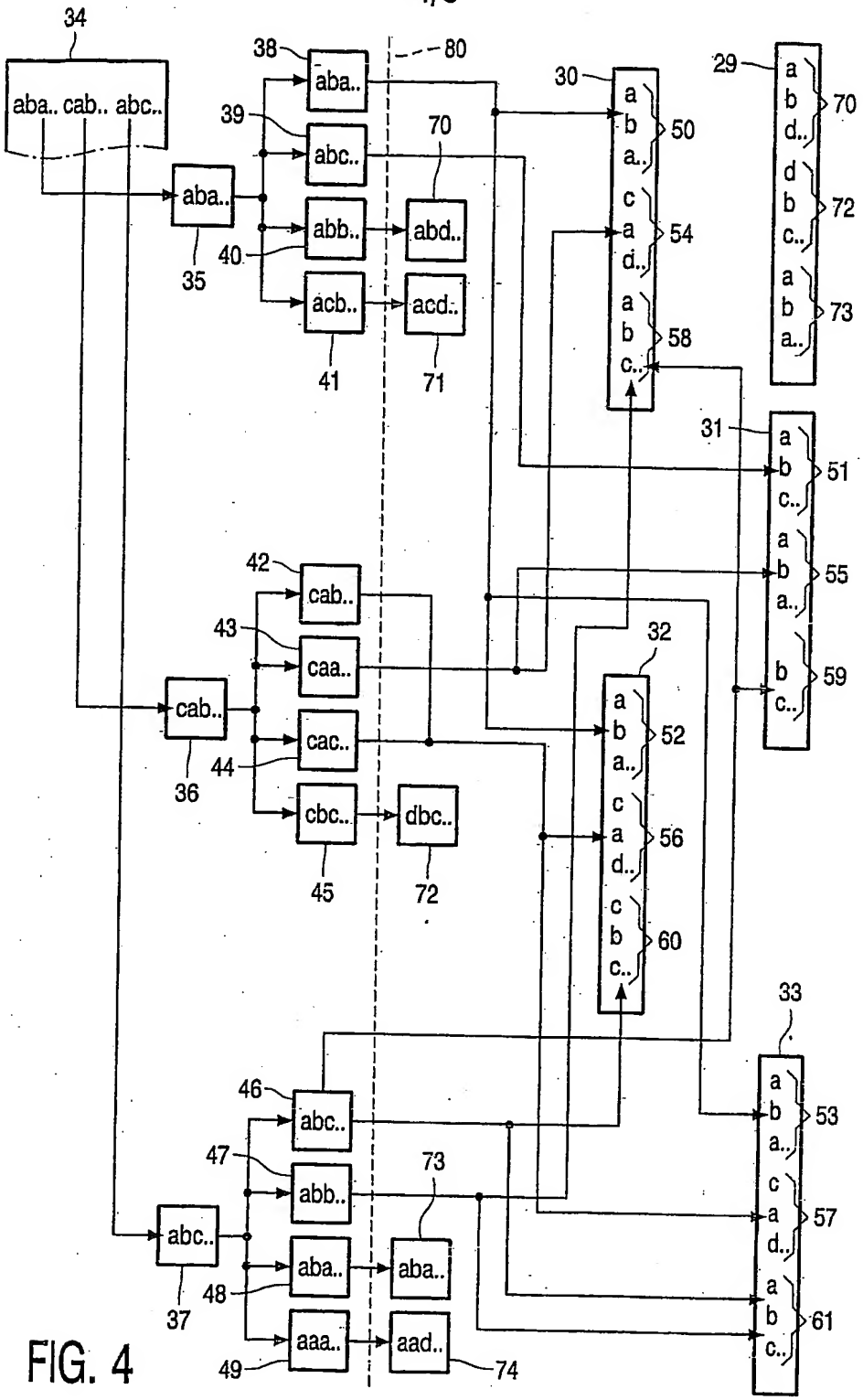

34
aba.. cab.. abc..
38
aba..
39
abc..
70
abd..
aba..
35
40
abb..
acb..
41
71
80
30
29
50
54
58
70
72
73
31
51
55
59
42
cab..
43
caa..
cab..
36
44
cac..
cbc..
dbc..
45
72
32
52
56
60
33
53
57
61
46
abc..
47
abb..
73
abc..
37
48
aba..
aba..
aaa..
aad..
49
74

FIG. 4

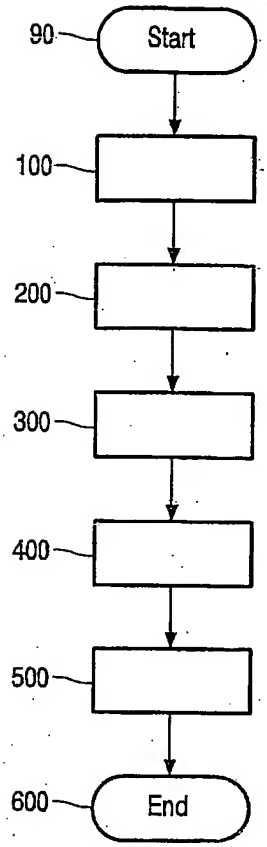



FIG. 5

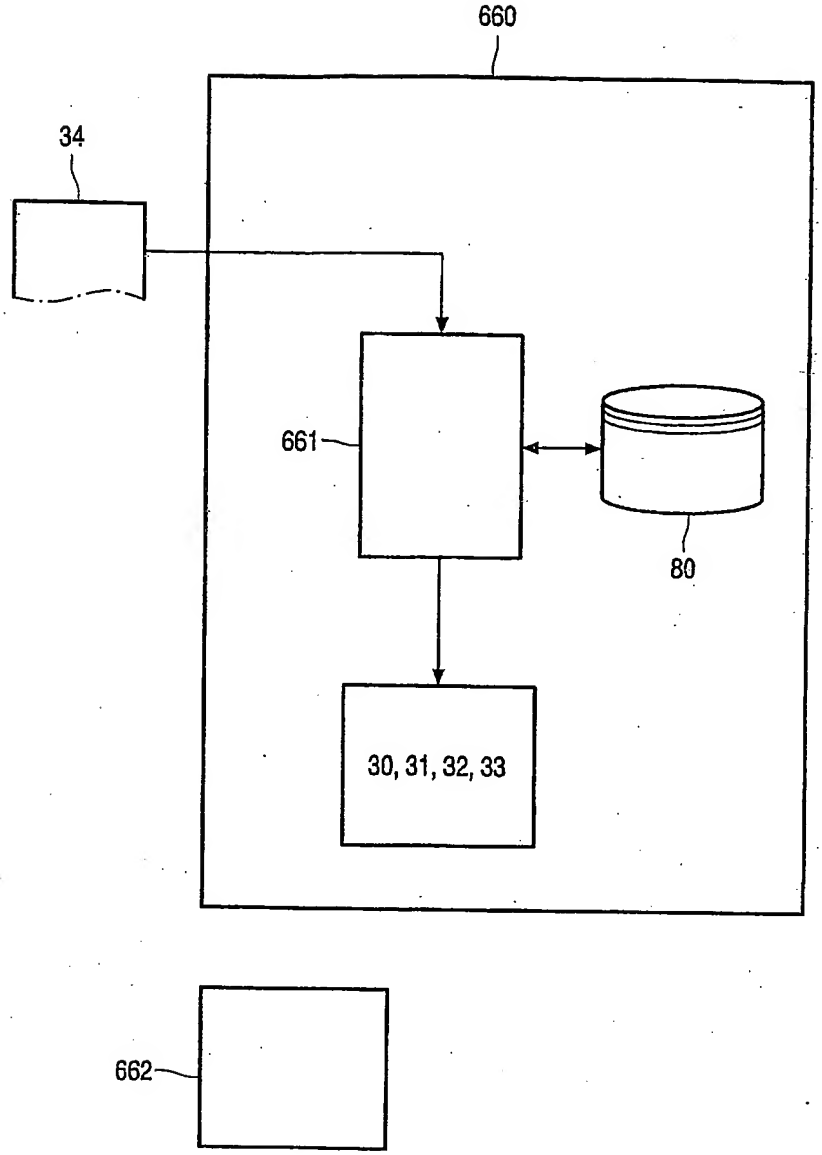



FIG. 6

(19)日本国特許庁(JP)　(12)公表特許公報(A)　(11)特許出願公表番号
特表2006-519445
(P2006-519445A)
(43)公表日 平成18年8月24日(2006.8.24)

| (51)Int.Cl. | F I | | テーマコード（参考） |
|---|---|---|---|
| G06F 17/30 (2006.01) | G06F 17/30 | 350C | 5B075 |

審査請求 未請求　予備審査請求 未請求　（全 19 頁）

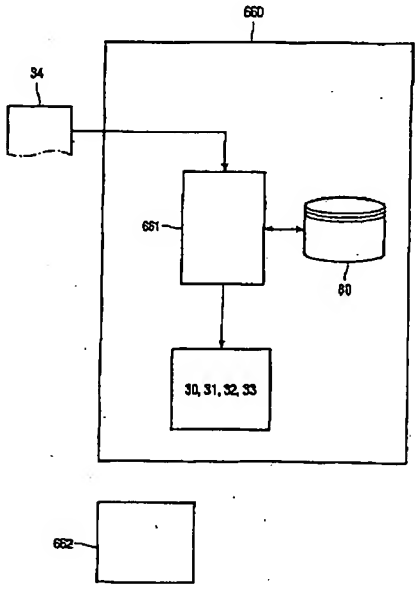
(21)出願番号　特願2006-506641 (P2006-506641)
(86)(22)出願日　平成16年2月25日 (2004.2.25)
(85)翻訳文提出日　平成17年9月1日 (2005.9.1)
(86)国際出願番号　PCT/IB2004/050148
(87)国際公開番号　WO2004/079631
(87)国際公開日　平成16年9月16日 (2004.9.16)
(31)優先権主張番号　03100517.6
(32)優先日　平成15年3月3日 (2003.3.3)
(33)優先権主張国　欧州特許庁 (EP)

(71)出願人　590000248
コーニンクレッカ　フィリップス　エレクトロニクス　エヌ　ヴィ
Ｋｏｎｉｎｋｌｉｊｋｅ　Ｐｈｉｌｉｐｓ　Ｅｌｅｃｔｒｏｎｉｃｓ　Ｎ．Ｖ．
オランダ国　５６２１　ベーアー　アインドーフェン　フルーネヴァウツウェッハ　１
Ｇｒｏｅｎｅｗｏｕｄｓｅｗｅｇ　１，５６２１　ＢＡ　Ｅｉｎｄｈｏｖｅｎ，Ｔｈｅ　Ｎｅｔｈｅｒｌａｎｄｓ
(74)代理人　100070150
弁理士　伊東　忠彦
(74)代理人　100091214
弁理士　大貫　進介

最終頁に続く

(54)【発明の名称】文字列検索の方法および設備

(57)【要約】
本発明は、多数の長い文字列を有する、あるいは単一の長い文字列を有するデータベース（８０）中における、問い合わせ文字列（３４）と部分一致または完全一致する内容をもつある最終的な個数の結果文字列（３０～３３）を検索する方法に関するものである。該方法は、問い合わせ文字列をある第一の個数の入力問い合わせ文字列（３５、３６、３７）に分割し、前記第一の個数の入力問い合わせ文字列のそれぞれの文字列に対してある第二の個数の近傍文字列（３８～４１、４２～４５、４４～４９）を決定し、ここで、前記第二の個数の近傍文字列のそれぞれの文字列は所定の第一の誤り個数の誤りを有するものとし、前記第二の個数の近傍文字列のそれぞれの文字列に対する完全一致文字列（５０～６１、７０～７４）を、ある検索方法に基づいて、ある第三の個数、データベースから検索し、前記データベースから検索された完全一致文字列をつなげてある第四の個数の中間文字列（２９、３０、３２、３３、３４）にし、ここで、前記中間文字列のそれぞれに含まれている検索された完全一致文字列（５０～６１、７０～７４）は前記デー



【特許請求の範囲】
【請求項１】
多数の長い文字列を有する、あるいは単一の長い文字列を有するデータベース中における、問い合わせ文字列と部分一致または完全一致する内容をもつある最終的な個数の結果文字列を検索する方法であって、
・前記問い合わせ文字列をある第一の個数の入力問い合わせ文字列に分割し、
・前記第一の個数の入力問い合わせ文字列のそれぞれの文字列に対してある第二の個数の近傍文字列を決定し、ここで、前記第二の個数の近傍文字列のそれぞれの文字列は所定の第一の個数の誤りを有するものとし、
・前記第二の個数の近傍文字列のそれぞれの文字列に対する完全一致文字列を、ある検索方法に基づいて、ある第三の個数、データベースから検索し、
・前記データベースから検索された完全一致文字列をつなげてある第四の個数の中間文字列にし、ここで、前記中間文字列のそれぞれに含まれている検索された完全一致文字列は前記データベース中で相続いているものとし、
・前記第四の個数の中間文字列に基づいて最終的な個数の結果文字列を決定し、ここで、前記最終的な個数の結果文字列のそれぞれの文字列は、前記問い合わせ文字列に比較して高々ある所定の第二の個数の誤りを有するようにする、
ステップを有することを特徴とする方法。
【請求項２】
前記検索方法がｑグラムインデックス法であることを特徴とする請求項１記載の方法。
【請求項３】
前記検索方法が接尾辞木法であることを特徴とする請求項１記載の方法。
【請求項４】
前記検索方法がハッシュ法であることを特徴とする請求項１記載の方法。
【請求項５】
前記文字列および前記データベースがそれぞれ西欧アルファベットの文字の列を有することを特徴とする、請求項１ないし４のうちいずれか一項記載の方法。
【請求項６】
前記文字列および前記データベースがそれぞれ楽譜の基本要素を表現することを特徴とする、請求項１ないし４のうちいずれか一項記載の方法。
【請求項７】
前記文字列および前記データベースがそれぞれ二進数字の列を有することを特徴とする、請求項１ないし４のうちいずれか一項記載の方法。
【請求項８】
前記文字列および前記データベースがそれぞれアミノ酸配列またはＤＮＡ／ＲＮＡ塩基配列を有することを特徴とする、請求項１ないし４のうちいずれか一項記載の方法。
【請求項９】
前記文字列および前記データベースがそれぞれビット、バイトまたは語の列を有することを特徴とする、請求項１ないし４のうちいずれか一項記載の方法。
【請求項１０】
請求項１ないし９のうちいずれか一項記載の方法のステップを実行する計算手段を有する検索エンジン。
【請求項１１】
請求項１ないし９のうちいずれか一項記載の方法のステップを実行する手段を有するツール。
【請求項１２】
・問い合わせ文字列をある第一の個数の入力問い合わせ文字列に分割する計算手段と、
・前記第一の個数の入力問い合わせ文字列のそれぞれの文字列に対してある第二の個数の近傍文字列を決定し、ここで、前記第二の個数の近傍文字列のそれぞれの文字列は所定の第一の個数の誤りを有するものとするような計算手段と、

・前記第二の個数の近傍文字列のそれぞれの文字列に対する完全一致文字列を、ある検索方法に基づいて、ある第三の個数、データベースから検索する計算手段と、
・前記データベースから検索された完全一致文字列をつなげてある第四の個数の中間文字列にし、ここで、前記中間文字列のそれぞれに含まれている前記検索された完全一致文字列は前記データベース中で相続いているものとするような計算手段と、
・前記第四の個数の中間文字列に基づいて最終的な個数の結果文字列を決定し、ここで、前記最終的な個数の結果文字列のそれぞれの文字列は、前記問い合わせ文字列に比較して高々ある所定の第二の個数の誤りを有するようにする計算手段とを有する、設備。
【請求項１３】
請求項１ないし９のうちいずれか一項記載の方法を実行するコンピュータシステム。
【請求項１４】
コンピュータ読み取り可能媒体上に記録されたプログラムコード手段を有し、コンピュータ上で実行されたときに請求項１ないし９のうちいずれか一項記載の方法を実行することを特徴とする、コンピュータプログラム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、多数の長い文字列を有する、あるいは単一の長い文字列を有するデータベース中における、問い合わせ文字列と部分一致または完全一致する内容をもつある最終的な個数の結果を検索する方法に関するものである。
【０００２】
本発明はまた検索エンジンにも関する。
【０００３】
本発明はまたツールにも関する。
【０００４】
本発明はまた前記方法を実行するコンピュータシステムにも関する。
【０００５】
本発明はさらに、前記方法を実行するコンピュータプログラムプロダクトにも関する。
【０００６】
加えて、本発明はさらに設備にも関する。
【背景技術】
【０００７】
米国特許第５，９６３，９５７号は音楽データベースを有する情報処理システムを開示している。前記音楽データベースは単旋律による音符の参照シーケンスを蓄えている。前記参照シーケンスはすべて辞書式に保存するために相対的な階名度数（ｓｃａｌｅ ｄｅｇｒｅｅ）に規格化されている。入力された音符列と特定の参照列との間の一致を見出すためにはＮ分木と呼ばれるものが適用される。これにより前記情報処理システムは一致する参照シーケンスに関連する書誌情報を提供するのである。
【０００８】
Du, D. W. and Chang, S. C. (1994) “An Approach to Designing Very Fast Approximate String Matching Algorithms” 〔非常に高速な近似文字列マッチングの諸アルゴリズム設計の試み〕, IEEE Transactions on Knowledge and Data Engineering, 6 (4), 620－633ではもう一つの種類の文字列マッチングがさらに開示されている。
【０００９】
当該技術では、検索方法は完全一致のアルゴリズムを用いる。既知の検索方法は典型的には完全一致を試みる。すなわち、完全に一致するものをみつけるために検索またはマッチングを行うのである。
【非特許文献１】Myers, E. (1994). A Sublinear Algorithm for Approximate Keyword Searching. Algorithmica, 12 (4/5), 345－374

【発明の開示】
【発明が解決しようとする課題】
【００１０】
しかし、多くの実用上の応用では、完全一致しか検索されないのは問題である。結果として、ちょっとした差異があるだけにも関わらず、役に立つかもしれない検索結果が与えられないというのは追加的な問題になる。
【００１１】
大きなデータベースの検索に長い時間がかかり、それに応じて計算機パワーの集中的使用が必要となることもさらなる問題である。
【００１２】
多くの実用上の応用では、（完全一致ではなく）部分一致を得るだけでも十分である。これは、問い合わせ文字列（検索試行への入力）もしくはマッチング結果の文字列またはその両方が比較的重要でない誤差を有していたとしても、全く結果が得られないよりは部分一致が得られるほうがましだからである。前記の誤差は、典型的には、問い合わせ文字列または検索されるデータベースに含まれる不適切なデータによって生じるものである。
【課題を解決するための手段】
【００１３】
上記のことを含むさまざまな問題が本発明の方法によって解決される。前記方法は以下のステップを有している。
・問い合わせ文字列をある第一の個数の入力問い合わせ文字列に分割する。
【００１４】
換言すれば、このステップでは前記問い合わせ文字列は前記第一の個数の小さな部分文字列片に、すなわち前記入力問い合わせ文字列に切り分けられる。
・前記第一の個数の入力問い合わせ文字列のそれぞれの文字列に対してある第二の個数の近傍文字列を決定する。ここで、前記第二の個数の近傍文字列のそれぞれの文字列は所定の第一の誤り個数の誤りを有している。
【００１５】
換言すれば、このステップでは近傍文字列の前記第二の個数は、前記問い合わせ文字列の長さ、使っている文字アルファベットにおける異なる区別される記号の数、そして前記近傍文字列において許される誤りの個数によって決まる。
【００１６】
一般に、前記第一の個数の入力問い合わせ文字列のそれぞれの文字列に対して、前記第二の個数の近傍文字列が決定される。そのそれぞれが個々に所定の第一の誤り個数の誤りを有しており、その数は０以上である。
・前記第二の個数の近傍文字列のそれぞれの文字列に対する完全一致文字列を、ある検索方法に基づいて、ある第三の個数、データベースから検索する。
【００１７】
ここで、前記第二の個数の近傍文字列のそれぞれの文字列に対して完全一致文字列を、ある所与の検索方法に基づいて、ある（第三の）個数、データベースから検索するときには、該検索方法はｑグラム（q−gram）法、接尾辞木（suffix tree）法、またはハッシュ法を使うことができる。
・前記データベースから検索された完全一致文字列をつなげてある第四の個数の中間文字列にする。ここで、前記中間文字列のそれぞれに含まれている検索された完全一致文字列は前記データベース中で相続いているものとする。
・前記第四の個数の中間文字列に基づいて最終的な個数の結果文字列を決定する。ここで、前記最終的な個数の結果文字列のそれぞれの文字列は、前記問い合わせ文字列に比較して高々ある所定の第二の誤り個数の誤りを有している。
【００１８】
最後の二つのステップについては、図５のステップ４００および５００において説明する。

【００１９】
前記方法の結果として、前記最終的な個数の結果文字列のそれぞれは、前記問い合わせ文字列の完全一致または部分一致になっている（冒頭の段落で言及）。
【００２０】
これにより、完全一致または軽微な誤りのみを含む部分一致を得ることが達成される。
【００２１】
さらに、前記方法は大きなデータベースも（完全一致または部分一致のために）比較的控えめな計算機パワーの使用によって高速に検索することができる。
【００２２】
前記設備、ツール、検索エンジン、コンピュータシステムはそれぞれ上に前記方法との関連で述べたのと同じ理由で同じ利点をもたらし、同じ問題を解決する。
【００２３】
本発明は以下において好ましい実施形態との関連で、また図面を参照しつつより詳細に説明される。
図面を通じて、同じ参照符号は同様なあるいは対応する特徴、機能、文字列などを示す。
【発明を実施するための最良の形態】
【００２４】
図１は本発明の技術の一般的な議論のための図である。この図は文字列データベースにおいて検索される文字列abacababc（参照符号８０）を示している。高々一つの誤りを許すとした場合の（k＝1）問い合わせ文字列の四つの近似一致文字列のデータベース中での位置が示されている。問い合わせ文字列（参照符号３４）のabacababcは３文字だけのアルファベット｛‘a’，‘b’，‘c’｝から構成されている。誤りが一つしか許されない場合（k＝1）、挿入された記号を含む近似一致（たとえば参照符号３０のabacababbc）、削除された記号を含む近時一致（たとえば参照符号３１のabcababc）、置換された記号を含む近似一致（たとえば参照符号３２のabacabcbc）、そして完全一致（すなわち、参照符号３３のabacababc）がみつけられる。当該技術において一般には、問い合わせ文字列全体がただちに検索されるような文字列の検索を行う仕方が認知されている。
【００２５】
図２は問い合わせ文字列を分割する様子を示している。文献から知られる効率的な検索諸方法は、完全一致検索、すなわち誤りを全く許さない（k＝0）検索のための高速アルゴリズムの使用を利用する。これは（完全）一致の場合の位置、すなわち図１における参照符号３３を返すだけである。高速な完全一致アルゴリズムの実装を可能にするためには、文字列データベース（参照符号８０）からオフラインで（プリプロセッサとして）接尾辞木またはｑグラムのような特殊なインデックス構造を構築しておく必要がある。本質的には、これらの構造は、データベース中に現れる小さな部分文字列の位置を一つ一つすべて保持しておくものである。事実上、これは検索プロセスが、データベース中の無関係な部分を無視して、関連する位置にすぐジャンプできることを意味している。ｑグラムインデックス法を使うことにするが、それは、他の方法よりスペース効率がよく（すなわち、メモリ使用量が少ない）、我々の目的に用意に適応させることができるからである。前記ｑグラムは、データベース中に現れる長さｑ＞０のあらゆる部分文字列の位置を保持している。もしたとえばデータベースが文字列abababcabcabacab．．．からなるものとすると、ｑグラムは長さ４のあらゆる部分文字列の開始位置を集めることになる。abab、baba、abab、babc、abca、といった具合である。これらの部分文字列は関数を使ってインデックス化され、アクセスが容易なデータ構造の形にリスト化され、ソートされる。今の例でいうababのような複数の同一の部分文字列は同じインデックス（バケツと呼ばれる）にたどりつく。ｑグラムによって、長さｍ≦ｑの問い合わせの完全一致についてはすべて、インデックス関数を計算してバケツの要素を取得することによってデータベース中の位置を得ることができる。ｑよりも長い問い合わせについては、問い合わせ文字列の長さｑのプレフィックスだけしかバケツ内にははいれないのであるから、さらなる検査が必要になる。ｑ

グラムの標準的な使い方はMyers (1994) に述べられている。

【００２６】

前記ｑグラム法の代わりに、接尾辞木法またはハッシュ法を適用してもよい。

【００２７】

完全一致法を使って近似マッチングを行う場合、誤差を許容するための工夫が必要になる。たとえば、本発明によれば、元来の問い合わせ（文字列）においてある限界以下の箇所で相違する文字列の組が生成できる。これらの文字列は、もとの問い合わせの近傍と呼ばれる。このような近傍が問い合わせにおける誤差を表す。厳密には、文字列Ｓのk近傍はＳに対して高々k個の誤りをもつ文字列の集合として定義される。たとえば、２文字だけのアルファベット｛‘a’，‘b’｝から文字列abbaの問い合わせを構成したとすると、高々一つの誤りをもつ（すなわち誤りレベルk≦1）近傍文字列の完全な集合は、元来のabba、削除を含むあらゆる文字列abb、aba、bba、挿入を含むあらゆる文字列aabba、babba、abbba、ababa、abbaa、abbab、そして置換を含むあらゆる文字列bbba、aaba、abaa、abbbとなる。

【００２８】

所与の文字列の近傍は効率的に生成することができる（Myers, 1994）。もしこれらの近傍がｑグラム法を使ってデータベース中に同定できたとしたら、それらの完全一致はもともとの問い合わせに対する近似一致に対応する。

【００２９】

しかしながら、調査すべき近傍文字列の数は調査する問い合わせ文字列が長くなり、アルファベット集合が大きくなり、誤りレベルを高くすると指数関数的に増大する。この問題を解決し、検索速度をかせぐため、問い合わせはまずより小さな部分文字列に分割され、各部分文字列について、その近傍文字列の集合が生成される。次いで、これらの近傍文字列すべてを、ｑグラムなり前述した他の検索方法なりを用いて完全一致により検索する。データベース中でのこれらの完全一致が今や、元来の問い合わせの部分的近似一致に対応するのである。

【００３０】

問い合わせ文字列を３文字だけのアルファベット｛‘a’，‘b’，‘c’｝から構成されるabacababcとし、誤りレベルは３まで許容する（k=3）ものとしよう。問い合わせにおける誤りはどの箇所にも生じうることを注意しておく。たとえば、誤りは次のようなものがありうる。

・全部が先頭部に（たとえば、これによるとccccababcが実際の検索でみつかる）
・全部が中部に（たとえばababbcabc）
・全部が末尾部に（たとえばabacabbca）
・問い合わせ文字列にまんべんなく分散（たとえばabccacabb）

問い合わせ文字列が３つの部分に分割されるとすると（p=3；この場合、今の例での問い合わせ文字列abacababcにおける部分文字列はaba、cab、abcとなる）、各部分について近傍文字列の集合が生成される。各近傍文字列は個別にデータベース中で検索され、元来の問い合わせがどのようなものであったかの前後関係は忘れられる。これを理解するために、近傍文字列はデータベース中のどこにでも生じうるものであることを注意しておく。近傍文字列どうしの現れ方は、元来の問い合わせの近似一致をなすためには必要とされるように近接または連続したものには、必ずしもならない。換言すれば、今の例の問い合わせ文字列abacababcについて、ある近傍文字列の完全一致が見つかったといっても、それが問い合わせ文字列の第一部分、第二部分、第三部分のどれに対応するものかも、他の二つの部分にどのような近傍文字列がみつかったかも知りようがないのである。この情報を明らかにするために、必要措置を講じなければならない。文献に記載されている以前の諸方法はまさにここでストップしていた。すなわち、ある近傍文字列の完全一致の一つ一つを問い合わせを解決するための有用な候補と見なしていたのである。それに対して、クロスカッティング（cross－cutting）の発明は、近傍文字列を検索する間に、（誤りの）前後関係を再現することによって追加的なフィルタステップを行うのである。破棄される近

傍文字列は次のようなものである。
・データベース中で他の近傍文字列との列をなして現れないもの。
・データベース中で他の近傍文字列と列をなしてはいるが、元来の問い合わせの近似一致ではありえないもの。
【００３１】
これらの観測は、本発明の中心となる「クロスカッティング」予備定理に凝縮される。これにより、問い合わせ文字列をp個の部分に分割してその各部分を高々$k_i$個の誤りで個別に検索する場合に、有意な部分がうまく引き出されることが保証されるのである。
【００３２】
クロスカッティング予備定理：AとBを２つの文字列とし、両者の間の相違の数は編集距離（edit−distance）の意味でk以下である、あるいは式で書いて$\partial(A, B) \leq k$とする。$A_i$を文字列として、任意のp>1について、$A=A_1A_2\cdots A_p$をAのp個の部分への分割とする。$K=(k_1, k_2, \cdots, k_p)$を任意の自然数の数列で
【００３３】
【数１】

$$C=\sum_{i=1}^{p}k_i+p-k\geq 1$$

となるものとする。このとき、
【００３４】
【数２】

$$\sum(k_{j_i}+1-\partial(A_{j_i},B_{j_i}))\geq C$$

となるようなある分割$B=B_1B_2\cdots B_p$および$J=(j_1, j_2, \cdots, j_t)$によって添え字が指定されるある部分集合が存在する。
【００３５】
証明：Bが$\Sigma\partial(A_i, B_i)=\partial(A, B)$〔和はi=1からpまで〕となるようにp個の部分に分割できることは明らかである。誤りは対応する部分に局在しており、新たな誤りが導入されることはないものとする。そのようなBの分割を選び、$k_i+1\geq\partial(A_i, B_i)$となるようなすべての部分iの部分集合Jを選ぶ。すると、次の不等式が成り立ち、それで証明が完了する。
【００３６】
【数３】

$$C=\sum_{i=1}^{p}k_i+p-k\leq\sum_{i=1}^{p}(k_i+1-\partial(A_i,B_i))\leq\sum_{i=1}^{t}(k_{j_i}+1-\partial(A_{j_i},B_{j_i}))$$

Aを問い合わせ文字列、Bをデータベース文字列とすると、この予備定理は本発明におけるフィルタリング条件として使われる。この予備定理は、データベース中の近傍文字列の列が表す誤りの数eがある特定の基準を満たさなければならないことを述べている。その基準として、問い合わせの各部分文字列iにおいて許される所定の誤り個数$k_i$を適用する。すると、まだ問い合わせ文字列の近似一致の範囲内であるためには、誤り総和$\Sigma(k_i+1)-e$〔和はi=1からpまで〕は少なくともある定数$C=\Sigma k_i+p-k$〔和はi=1からpまで

] 以上であるべきだというのである。これらの公式において、pは問い合わせ文字列が分割される部分文字列の数、$k_i$は各部分文字列iに許される誤りの数、kは誤りの総数の最大値（誤りレベル）である。

【００３７】

誤り総和Σ（$k_i+1$）－e〔和はi＝1からpまで〕を計算し、それをCと比較することが前記クロスカッティング・アルゴリズムの基礎である。一言で言えば、データベース中のある特定の位置にある近傍文字列に対する新たな一致がみつかるたびに、そのデータベース位置の前に他の近傍文字列の一致がないかどうか検査される。特殊なデータ構造により、データベース中の近傍文字列のすべての位置は効率的な仕方で把握されている。そうして、（データベース中に現れる）これらの連続した一致文字列をつなげたものが最終的に完全な問い合わせ文字列の近似一致になりうるかどうかが検証される。もし誤り総和が閾値C以上であれば、これらの近傍文字列は、いまだに問い合わせの近似一致の範囲内の有意な候補である。もし誤り総和がこの閾値C未満であれば、関連する近傍文字列はみな破棄される。

【００３８】

図２に示すように、問い合わせ文字列abacababc（参照符号３４）は３つの部分文字列に分割される（p=3）。許される誤りが３つだけである（k=3）ことを想起し、$k_i$＝floor（k／p）＝１〔floorは床関数〕およびC＝Σ$k_i$＋p－k＝3〔和はi＝1からpまで〕と定義する。各部分文字列に対して、近傍文字列の集合（すなわち、それぞれ参照符号３８～４１、４２～４５、４６～４９）が生成され、その文字列が完全一致によってデータベース中で検索される。この近傍文字列検索の過程では、それまでにみつかった全近傍文字列の位置が保持され、連続する近傍文字列をつなげたものが問い合わせの近似一致の一部となりうるかどうかの判定がなされる。近傍文字列abaおよびcabの二つの一致（図２の参照符号３０を参照）は、その問い合わせ文字列の最初の二つの部分文字列についての誤りのない一致を表している（すなわち、e=0でΣ（$k_i+1$）－e=4≧C=3〔和はi＝1から2まで〕）。これですでに問い合わせに対する有効な近似一致がみつかったことがわかる。次にくる近傍文字列が３つ誤りを含むという最悪の場合でも、我々のフィルタリング条件はまだ成立するのである。近傍文字列abcおよびcaaに対する二つの一致（図２の３１参照）は二つの誤りがある場合を表している（すなわち、e=2でΣ（$k_i+1$）－e=2〔和はi＝1から2まで〕）。この連続は、問い合わせの有効な近似一致にとどまるためには、次に来る近傍文字列の誤りは高々１つでなければならない。近傍文字列acbおよびcbcに対する一致（図２の参照符号３２を参照）はすでに４つの誤りを含んでいる（すなわち、e=4でΣ（$k_i+1$）－e=0〔和はi＝1から2まで〕）。これですでにこの近傍文字列の列は問い合わせの近似一致の部分とはなりえないことがわかる。たとえ次に誤りのない近傍文字列がきたとしても、フィルタリング条件は満たされないのである。

【００３９】

前記ｑグラムおよび前記近傍生成についてのより詳細で全般的な議論として、以下の節により当業者は本発明を実施することができるであろう。

ｑグラム、すなわちｑグラムインデックス法

ｑグラムを用いれば、ｑを超えない長さの文字列の生起箇所すべてを非常に高速にみつけることが可能である。これらのｑグラムは次のようにして構成される。

【００４０】

Σ中の記号の整数０から$\sigma-1$への全単射$\phi$を考える。関数$\phi$は漸化式$\phi(Pa)=\sigma\phi(P)+\phi(a)$によって自然に文字列に拡張できる。ここで、Pはは Σ上の文字列、aはΣに含まれる記号である。b＝〔0，$\sigma^q-1$〕に対して、Bucket（b）＝｜i：$\phi(a_i a_{i+1}\cdots a_{i+q-1})=b$｜とする。すなわち、Bucket（b）は$\phi$値がbであるようなq個の記号からなる一個の文字列のA内における各生起例の左端の記号の添え字を与える関数である。

【００４１】

前記添え字は次のようにして生成される。まず、$\phi i=\phi(a_i a_{i+1}\cdots a_{i+m-1})$があら

ゆる添え字iについて計算される。これは、$\phi i = a_i \sigma^{m-1} + \mathrm{floor}(\phi_{i+1} / \sigma)$であることを利用すれば、Aをなめる$O(n)$の操作によって実行できる。今度は、$O(n\log(n))$のクイックソートを用いて$\phi_{I[j]} \leq \phi_{I[j+1]}$となるようなリストIndices$=\langle i_1, i_2, \cdots, i_n \rangle$が生成できる。最後に、配列Header $[b] = \min\{j : \phi_{Indices[j]} = b\}$がIndicesをなめる$O(n)$の操作によって生成される。配列IndicesおよびHeaderがBucketの集合を実現したものとなる。すなわち、Bucket $[b] = \{Indices[j] : j \in [Header[b], Header[b+1] - 1]\}$となる。

【００４２】

問い合わせ文字列Pの長さが$m \leq q$とすると、Pの生起箇所を表す添え字の全体はちょうど$b \in [\phi(P)\ \sigma^{q-m},\ (\phi(P)+1)\ \sigma^{q-m}-1]$に対するBucket(b)の中身になる。

【００４３】

問い合わせ文字列Pの長さがqを超えている場合には、Pの生起箇所の集合はBucket$(\phi(P_q))$の部分集合であることがわかる。ここで、$P_q$はPの最初のq個の記号からなる文字列を表す。

近傍の生成

文字列Ｐの（完全な）k近傍はＰからの（編集）距離がk以下であるすべての文字列の集合として定義できる。すなわち、$N_k(P) = \{Q : \partial(Q, P) \leq k\}$。

【００４４】

文字列Ｐの凝縮（condensed）k近傍はＰの完全なk近傍に属するすべての文字列のうち当該近傍にプレフィックスをもたないものの集合である。すなわち、$C_k(P) = \{Q : Q \in N_k(P)$ かつQは$N_k(P)$内にプレフィックスをもたない$\}$。

【００４５】

マイヤースのアルゴリズムは文字列の凝縮k近傍を効率的に計算する。それは、アルファベット集合から語を生成し、現在生成されている語とＰのダイナミックプログラミング行列（dynamic programming matrix）の対応する列を計算することによる。ある語が現在の列の最後のエントリーがkに等しければ、凝縮k近傍における語に到達したことになる。もしすべてのエントリーがkより大きければ、アルゴリズムはもとに戻ることができる。失敗リンクの使用により、このアルゴリズムはk近傍にありながら凝縮k近傍には含まれない語を逃すことを防いでいる。

【００４６】

本発明は、データベース中から分割文字列のあらゆる完全一致をみつけるために文字列の完全なk近傍を必要としているので、マイヤースのアルゴリズムは修正して使われる。

【００４７】

図３は問い合わせ文字列の分割とその後の検索の様子を実際上の詳細な例で示している。

【００４８】

問い合わせ文字列（参照符号３４）が再びデータベース（参照符号８０）中で検索される。本発明によれば、前記問い合わせ文字列はいくつかの入力問い合わせ文字列に分割される。ここでは簡単のためその数を３としているが、１より大きないくつでもよい。今の例では、前記入力問い合わせ文字列の先頭部、中部、末尾部がそれぞれ参照符号３５、３６、３７によって表されている。

【００４９】

前記の数の入力問い合わせ文字列によって、いくつかの近傍文字列（ここでは４つとする）が入力問い合わせ文字列のそれぞれに対して定義される。すなわち、参照符号３５の入力問い合わせ文字列に対しては、対応する４つの近傍文字列（参照符号３８、３９、４０、４１）が定義される。

【００５０】

同様にして、参照符号３６の「中部」入力問い合わせ文字列に対しては、対応する４つの近傍文字列（参照符号４２、４３、４４、４５）が定義される。

【００５１】
同様にして、参照符号３７の「末尾部」入力問い合わせ文字列に対しては、対応する４つの近傍文字列（参照符号４６、４７、４８、４９）が定義される。
【００５２】
破線の右側（参照符号８０）では、図のこの部分においてはデータベース（以前にも同じ参照符号で示されていた）が検索されていることが含意されている。すなわち、前記近傍文字列（参照符号３８～４９）のそれぞれが（部分文字列の）完全一致をみつけるために検索されるのである。
【００５３】
これらは矢印をさらに右にたどっていくことにより示される。例を挙げると、先頭部の近傍文字列である参照符号３８は参照符号５０の完全一致を与える。別の例を挙げると、末尾部の近傍文字列である参照符号４７は参照符号５８および６１の一致を与える。中部の近傍文字列である参照符号４５は参照符号７２の「使えない」結果を与える。
【００５４】
そして多かれ少なかれ問い合わせ文字列（参照符号３４）に一致する最終的な検索結果を達成するために、さらに矢印を右にたどる。すなわち、参照符号３０～３３はそれぞれ四つの最終的な検索結果の一つを示している。図からわかるように、前記最終的な検索結果のそれぞれは、必ず、検索された先頭部の部分文字列（参照符号５０～５３）の一つ、検索された中部の部分文字列（参照符号５４～５７）の一つ、検索された「末尾部」の部分文字列（参照符号５８～６１）の一つからなる。これらがどのように相続いて配されるか、そしてその基準については、のちに図５を用いて説明する。
【００５５】
図４は、問い合わせ文字列を分割、検索する様子を一般的な場合の例で示す図である。図４は図３に対応しているが、全体的に「．．」によってどの参照符号のどの文字列も構成文字数がより少なかったり多かったりしてもよいことを示している。すなわち、本発明は非常に短い文字列にも、非常に長い文字の系列にも同じように適用できるのである。
【００５６】
図示したような西欧アルファベットの文字列の代わりに、音高アルファベットの要素の列、音程アルファベットの要素の列、音長アルファベットの要素の列、二進の数字、語、バイトの列、アミノ酸の配列やＤＮＡ／ＲＮＡの塩基配列でもよい。これに対応して、同じことは、検索されるデータベースについてもあてはまる。データベースのほうも一つの長い文字列とも、多くの長い文字列とも考えられるのである。
【００５７】
前記の音程アルファベットの要素の列および音長アルファベットの要素の列が楽譜の基本要素をなすものである。一般に、すべての文字列について（問い合わせ文字列、データベース文字列など）、区別できる記号からなるいかなるアルファベット集合から構成してもよいということである。
【００５８】
図５は最終的な個数の結果文字列を検索する方法を示している。該方法は（これまでの図の）参照符号３０～３３によって示される最終的な個数の結果文字列の検索をする。すなわち、前記最終的な個数の結果文字列のそれぞれは、データベース中で問い合わせ文字列（参照符号３４）と部分一致またはできれば完全一致したものである。データベース（参照符号８０）は一つの長い文字列からなる。前記方法は以下のステップを有する。
【００５９】
ステップ１００では、問い合わせ文字列がある第一の個数の入力問い合わせ文字列に分割される。これまでの図で示したところでは、前記問い合わせ文字列は３つの入力問い合わせ文字列（参照符号３５、３６、３７）に分割される。すなわち、前記第一の個数はここでは３である。前記第一の個数は１以上のいかなる数であってもよい。前記第一の個数をちょうど３としたのは、解説の目的のためにすぎず、これより大きかったり小さかったりするいかなる数を選んでもかまわない。

【００６０】
換言すれば、このステップでは前記問い合わせ文字列は（前記第一の個数の）小さな部分文字列片に、すなわち前記入力問い合わせ文字列に切り分けられる。
【００６１】
今の例では、問い合わせ文字列（参照符号３４）aba. . cab. . abc. . が前記第一の個数の入力問い合わせ文字列の組に切り分けられる。今の例では一つの組には３つがある。すなわち、入力問い合わせ文字列１（参照符号３５）aba. . 、入力問い合わせ文字列２（参照符号３６）cab. . 、入力問い合わせ文字列３abc. . である。
【００６２】
ステップ２００では、ある第二の個数の近傍文字列が決定される。やはりこれまでの図で示したところでは、近傍文字列の前記第二の個数は４である。すなわち、最初の入力問い合わせ文字列（参照符号３５）には参照符号３８～４１、第二の、すなわち中部の入力問い合わせ文字列（参照符号３６）には参照符号４２～４５、第三の、すなわち最後の入力問い合わせ文字列（参照符号３７）には参照符号４６～４９である。
【００６３】
前記第二の個数をちょうど４としたのは、解説の目的のためにすぎず、これより大きかったり小さかったりするいかなる数を選んでもかまわない。特に、近傍文字列の数は問い合わせ文字列の長さ、使われている文字列用アルファベット集合に含まれる異なる区別できる記号の数、近傍文字列中で許される誤りの個数に依存する。
【００６４】
これで今の例では合計１２の近傍文字列が生じる。すなわち、前記第一の個数かける前記第二の個数すなわち３×４＝１２、すなわち（３つの）入力問い合わせ文字列それぞれに４つずつである。一般には、前記第一の個数の入力問い合わせ文字列のそれぞれに前記第二の個数ずつの近傍文字列が決定される。これまでの図で示したところでは、これらは３８～４９にあたる。これらのそれぞれは、個々に所定の第一の誤り個数の誤りを含み、該第一の誤り個数は０以上である。
【００６５】
ただし、もしも（第一の）誤り個数が近傍文字列の長さを超えたら（すなわち、当該文字列の全内容が誤っているように決められるようになる）、次のステップにおける続く検索は完全に無意味になる。よって、前記第一の誤り個数は前記文字列長を超えることはできない。
【００６６】
例を挙げると、入力問い合わせ文字列aba. .（参照符号３５）をもとに４つの近傍文字列が決定される。すなわち、
・入力問い合わせ文字列自身に等しい、すなわちもちろん誤りのないaba. .（参照符号３８）
・１個の誤りを含むabc. .（参照符号３９）
・１個の誤りを含むもう一つのabb. .（参照符号４０）
・２個の誤りを含むacb. .（参照符号４１）
挙げられている例では、前記所定の第一の誤り個数（０以上の数）はここでは０、１、または２である。
【００６７】
前記所定の第一の誤り個数を今の例で０、１、または２としたのは解説の目的のためにすぎず、これより大きいいかなる数を選んでもかまわない。
【００６８】
ステップ３００では、前記第二の個数の近傍文字列の各文字列の完全一致をデータベース中である第三の個数検索する。前記検索はある所与の検索方法に基づいて行われる。
【００６９】
前記第三の個数の完全一致は参照符号５０～６１および７０～７４によって図示されている。ここで一致が一つまたはそれ以上ありうることを留意しておくことが重要である。

たとえば、
・第一に、近傍文字列の例aba. . （参照符号３８）は参照符号５０および５２、すなわちaba. . の完全一致につながる。
・第二に、近傍文字列のもう一つの例abc. . （参照符号３９）はやはり参照符号５１との完全一致につながる。
・第三に、abb. . （参照符号４０）は一致にはつながらない。すなわち参照符号７０のabd. .
・最後に、acb. . （参照符号４１）も一致にはつながらない。すなわち参照符号７１のabc. .
最後の二つは完全一致のみを考えているので使うことができない。

【００７０】
同様にして、参照符号５３～６１および７２～７４も参照符号４２～４９によって示されている近傍文字列からの検索結果である。

【００７１】
どの場合でも、検索結果（参照符号５０～６１および７０～７４）は対応する文字列内容とデータベース中の対応する位置とともにのちに後続のステップで使うために保持される。

【００７２】
また、どの場合でも、前記した所与の検索方法とは、ｑグラムインデックス法でも当業界において有用であると知られている他のいかなる好適な方法、たとえば接尾辞木法やハッシュ法でもよい。

【００７３】
ステップ４００では、データベースから検索された前記完全一致文字列をつないである第四の個数の中間文字列がつくられる。前記検索された完全一致文字列は（前記中間文字列のそれぞれに取り込まれるときは）前記データベース中で相前後して存在する。

【００７４】
前記第四の個数の中間文字列は参照符号２９～３３で示されている。示されている例での前記第四の個数は５である。さらに、前記中間文字列のそれぞれに含まれる前記検索された完全一致文字列（参照符号５０～６１および７０～７４によって示される）は、前記データベースにおいて相前後して存在するよう決定される。これについて以下に説明する。

【００７５】
諸例から見て取れるように、つなぐ際、最初の入力問い合わせ文字列（問い合わせ文字列の先頭部である）aba. . （参照符号３５）は対応する先頭部の近傍文字列（参照符号３８～４１）を有し、対応する先頭部の部分文字列（参照符号５０～３３）を導く。

【００７６】
同様にして、つなぐ際、第二の入力問い合わせ文字列（問い合わせ文字列の中部である）cab. . （参照符号３６）は対応する「中部」の近傍文字列（参照符号４２～４５）を有し、対応する中部の部分文字列（参照符号５４～５７）を導く。

【００７７】
同様にして、つなぐ際、第三の入力問い合わせ文字列（問い合わせ文字列の末尾部である）abc. . （参照符号３７）は対応する「末尾部」の近傍文字列（参照符号４６～４９）を有し、対応する「末尾部」の部分文字列（参照符号５８～６１）を導く。

【００７８】
換言すれば、前記中間文字列のそれぞれの対応する部分をなす完全一致した文字列（参照符号５０～６１、７０～７４）が実はデータベース中で相前後して存在する。すなわち、先頭部文字列に対応するもの（参照符号５０～５３）、中部文字列に対応するもの（参照符号５４～５７）、末尾部文字列に対応するもの（参照符号５８～６１）がつなげられて前記第四の個数の中間文字列（参照符号２９～３３）の一つをなすのである。

【００７９】

ステップ５００では、最終的な個数の結果文字列が決定される。その決定は、先行ステップからの前記第四の個数の中間文字列に基づいて行われるもので、ここで―このステップにおいて―前記最終的な個数の結果文字列の文字列のそれぞれが、当該問い合わせ文字列（参照符号３４）に比較しての誤りが高々ある所定の第二の誤り個数となるように決定される。

【００８０】

挙げられている例では、結果文字列（参照符号３０～３３）の前記最終的な個数は４である。一方、中間文字列の前記第四の個数は５であった。すなわち、今の例では、参照符号２９に相当する一つが破棄または無視されているのである。それは、これが特に、誤りが前記第二の誤り個数以下とする基準を満たさないからである。これは当該問い合わせ文字列（参照符号３４）と比較してみると見て取れる。

【００８１】

換言すると、参照符号２９は（最初の問い合わせ文字列（参照符号３４）と比較したときの）誤りが多すぎるために破棄される。一方、参照符号３０～３３はいずれもそれより誤りが少なく、前記基準を満たしていたのである。つまり、今の例では参照符号３０～３３が最終的な個数の結果文字列をなす。

【００８２】

本方法の結果として、前記最終的な個数の結果文字列（参照符号３０～３３）のそれぞれは当該問い合わせ文字列（参照符号３４）の完全一致または部分一致である。

【００８３】

挙げられている例では、こうして４つの一致文字列（完全一致または部分一致）が、当該問い合わせ文字列を検索したときの結果となる。

【００８４】

このステップは先のステップと合わせて「クロスカッティング」とも呼ばれる。これはすなわち、（検索されたときの）近傍文字列の完全一致のうちでも、つないだときに元来の問い合わせ文字列（参照符号３４）との近似一致を含みうるもののみを考えるという発想である。

【００８５】

本発明の精神では、前記「第一の個数」「近傍文字列の第二の個数」「第三の個数」「中間文字列の第四の個数」「第一の誤り個数」「第二の誤り個数」は個別に、あるいは相互の関連で、あるいは問い合わせ文字列もしくはデータベースまたはその両方の内容との関連で微調整してもよい。これにより検索速度を加減したり、一致度の異なる（より少ない）誤りを得たりすることができる。

【００８６】

同様に、挙げられている例は解説のためのもので、問い合わせ文字列、近傍文字列、中間文字列の長さが違ったり、文字列に含まれる連なりの内容（区別できる記号）が違っていたりなどする場合に拡張することもできる。

【００８７】

図６は検索のための設備を示す。参照符号６６０は前記設備を示す。該設備は先の図で議論したように、本発明に基づいて問い合わせ文字列（参照符号３４）を処理する。該設備は前記文字列を入力として処理し、そのため計算手段６６１、たとえば十分高速なマイクロプロセッサを有している。該マイクロプロセッサはデータベース（参照符号８０）中で一致するものを検索する。結果として、もしあれば最終的な個数の結果文字列（参照符号３０、３１、３２、３３）がみつけられる。マッチング方法のステップを実行する計算手段はまた、たとえば専用ＡＳＩＣであってもよい。

【００８８】

参照符号６６２はコンピュータプログラムプロダクトを表す。前記コンピュータプログラムプロダクトはコンピュータ読み取り可能媒体上に記録されたプログラムコード手段を有し、該コンピュータプログラムがコンピュータ上で実行されたときに前記方法を実行する。

【産業上の利用可能性】
【００８９】
一般に、本発明は音楽システム用旋律検索（「ハミングによる検索」）、検索エンジンやテキストファイルにおけるキーワード検索、分子生物学のデータベース中でのＤＮＡ／ＲＮＡ塩基配列やビット、バイト、語のコードの検索、誤り制御など、さまざまな分野において応用可能である。旋律検索の応用については、メロディーのある小さな断片だけ覚えていてメロディーまたは歌の全体は覚えていない場合を考えることができる。区別できる記号の列として適切な表現形式でひとたび与えられれば、このメロディー断片が検索方法に入力され、前記データベースを使って歌や旋律の素性を明らかにする。前記設備はたとえばジュークボックス―単体のオーディオ装置でもパソコン上に実装されたものでも―でもよい。あるいは携帯オーディオ装置であって、たとえばジョギング中の人が出だしかさびを口笛で吹くことによって伴奏音楽をすばやく変えることができるようになるインターフェースを含んだものでもよい。前記設備はまた、たとえばインターネットサーバー上でウェブから特定のＭＰ３をすばやく選択するためのサービスであってもよい。あるいは前記設備は携帯型電話であって、前記方法をたとえば着信メロディーを検索するために走らせているものでもよい。
【００９０】
同様に、検索エンジンへの問い合わせとしてのキーワード（前述した問い合わせ文字列と同様のもの）―たとえばインターネット上で特定の製品を検索したり、ソフトウェア辞書やソフトウェア検索ツールにおいてある単語を検索したりするためのもの―に入力ミスが含まれていてもよくなる。検索系がそうしたキーワード中の誤りに対処できる。あらゆる応用分野において、許容される誤りの数はあらかじめ定義され、固定されているのでもよい。データベースは一つのきわめて長大な文字列（たとえば、世界中のあらゆる旋律を一列につないだ長大なテキストなど）と見なすのが最もよい。また、文字列はどんな有限集合（たとえば、西欧アルファベット、音高アルファベット、二進の数字、バイト、語、アミノ酸、ＤＮＡ／ＲＮＡ、語など）からでも構成することができる。テキストでの応用に関しては、西欧アルファベットの２６文字を使うことができる。同様に、旋律は９要素からなる音程アルファベットから構成することができる。分子生物学分野では２０のアミノ酸または４つのヌクレオチドをアルファベットとして用いる。プログラミング分野では二進の記号、語、ビット、バイトを使う。
【００９１】
楽譜の基本要素というもとに、旋律を検索するのに十分なあらゆる情報が含まれるものと理解される。たとえば、音程でもよいし、あるいは利用者があまり音楽的でなかったり、たとえば足を踏み鳴らして楽曲を検索したかったり、あるいはその両方の場合には単に音長だけでもよいし、あるいは音程と音長の両方でもよい。これらが所定の対応付け関数によって文字列をなす記号に変換されるのである。
【００９２】
コンピュータ読み取り可能媒体は磁気テープ、光ディスク、デジタル多用途ディスク（ＤＶＤ）、コンパクトディスク（記録可能ＣＤまたは書き込み可能ＣＤ）、ミニディスク、ハードディスク、フロッピー（登録商標）ディスク、ＩＣカード、ＰＣＭＣＩＡカードなどであることができる。
【００９３】
特許請求の範囲において、括弧に入れて参照符号があったとしても、それは特許請求の範囲を限定するものと解釈してはならない。「有する」の語は請求項において挙げられている以外の要素やステップの存在を排除するものではない。要素の単数形の表現はそのような要素が複数存在することを排除するものではない。
【００９４】
本発明はいくつかの明確に区別される要素を有するハードウェアによって、また好適にプログラミングされたコンピュータによって実装されることができる。いくつかの手段を列挙している装置請求項において、こうした手段のいくつかが単一のハードウェア要素に

よって具体化されることも可能である。ある複数の方策が互いに異なる従属請求項において述べられているというだけのことをもってそれらの方策の組み合わせが有利に利用できる可能性を排除していることにはならない。

【図面の簡単な説明】

【００９５】

【図１】本技術の一般的な議論のための図である。

【図２】問い合わせ文字列を分割する様子を示す図である。

【図３】問い合わせ文字列の分割とその後の検索の様子を実際上の詳細な例で示す図である。

【図４】問い合わせ文字列を分割、検索する様子を一般的な場合の例で示す図である。

【図５】最終的な個数の結果文字列を検索する方法を示す図である。

【図６】検索のための設備を示す図である。

【図１】

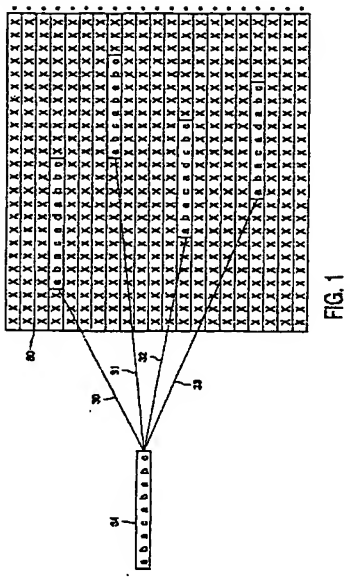


FIG. 1

【図２】

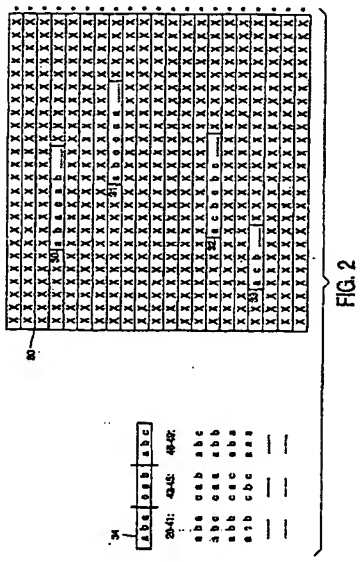


FIG. 2

【図３】

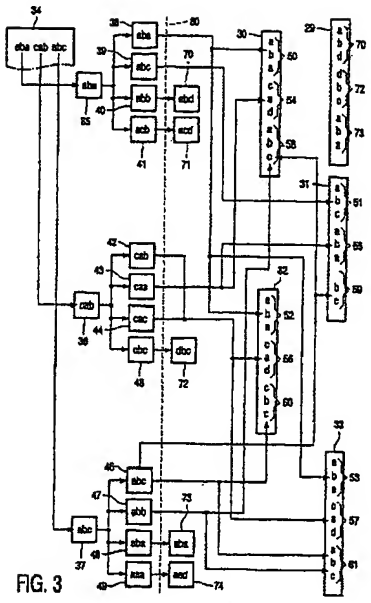


FIG. 3

【図４】

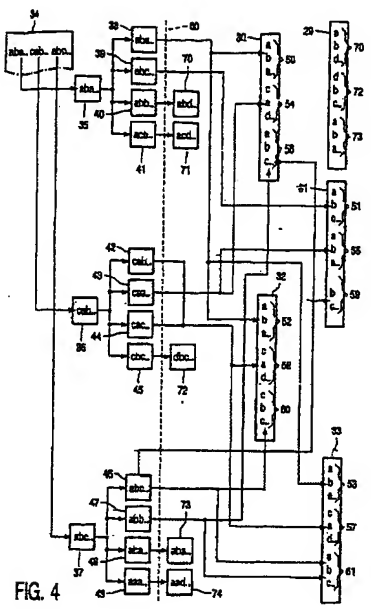


FIG. 4

【図５】

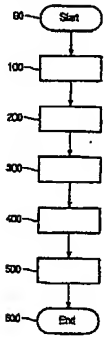



FIG. 5

【図６】

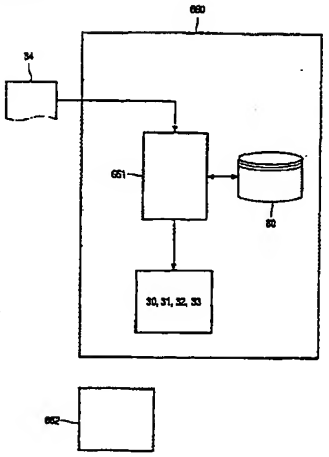



FIG. 6

【国際調査報告】

## INTERNATIONAL SEARCH REPORT

International Application No
PCT/IB2004/050148

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F17/30

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, WPI Data, INSPEC

C. DOCUMENTS CONSIDERED TO BE RELEVANT

| Category * | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| X | MYERS E. W.: "A Sublinear Algorithm for approximate keyword searching" ALGORITHMICA, vol. 12, no. 4-5, October 1994 (1994-10), pages 345-374, XP008033755 GERMANY cited in the application the whole document<br>-/-- | 1-14 |

[X] Further documents are listed in the continuation of box C. [ ] Patent family members are listed in annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier document but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
"&" document member of the same patent family

| Date of the actual completion of the international search | Date of mailing of the international search report |
|---|---|
| 9 August 2004 | 01/09/2004 |

| Name and mailing address of the ISA | Authorized officer |
|---|---|
| European Patent Office, P.B. 5818 Patentlaan 2<br>NL - 2280 HV Rijswijk<br>Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,<br>Fax: (+31-70) 340-3016. | DE CASTRO PALOMARES |

Form PCT/ISA/210 (second sheet) (January 2004)

## INTERNATIONAL SEARCH REPORT

International application No
PCT/IB2004/050148

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

| Category * | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| A | LUIS GRAVANO AND OTHERS: "Using q-grams in a DBMS for Approximate String Processing" IEEE DATA ENGINEERING BULLETIN, 'Online! vol. 24, no. 4, 2001, pages 28-34, XP002291636 Retrieved from the Internet: URL:http://citeseer.ist.psu.edu/cache/papers/cs/27618/http:zSzzSzwww1.cs.columbia.eduzSz{pirotzSzpublicationszSzdeb-dec2001.pdf/gravano01using.pdf> 'retrieved on 2004-08-06! the whole document | 1-14 |

Form PCT/ISA/210 (continuation of second sheet) (January 2004)

フロントページの続き

(81)指定国 AP(BW,GH,GM,KE,LS,MW,MZ,SD,SL,SZ,TZ,UG,ZM,ZW),EA(AM,AZ,BY,KG,KZ,MD,RU,TJ,TM),EP(AT,BE,BG,CH,CY,CZ,DE,DK,EE,ES,FI,FR,GB,GR,HU,IE,IT,LU,MC,NL,PT,RO,SE,SI,SK,TR),OA(BF,BJ,CF,CG,CI,CM,GA,GN,GQ,GW,ML,MR,NE,SN,TD,TG),AE,AG,AL,AM,AT,AU,AZ,BA,BB,BG,BR,BW,BY,BZ,CA,CH,CN,CO,CR,CU,CZ,DE,DK,DM,DZ,EC,EE,EG,ES,FI,GB,GD,GE,GH,GM,HR,HU,ID,IL,IN,IS,JP,KE,KG,KP,KR,KZ,LC,LK,LR,LS,LT,LU,LV,MA,MD,MG,MK,MN,MW,MX,MZ,NA,NI,NO,NZ,OM,PG,PH,PL,PT,RO,RU,SC,SD,SE,SG,SK,SL,SY,TJ,TM,TN,TR,TT,TZ,UA,UG,US,UZ,VC,VN,YU,ZA,ZM,ZW

(74)代理人 100107766
弁理士 伊東 忠重
(72)発明者 エグナー，セバスティアン
オランダ国，５６５６ アーアー アインドーフェン，プロフ・ホルストラーン ６
(72)発明者 コルスト，ヨハネス ハー エム
オランダ国，５６５６ アーアー アインドーフェン，プロフ・ホルストラーン ６
(72)発明者 ファン フューレン，マルセル
オランダ国，５６５６ アーアー アインドーフェン，プロフ・ホルストラーン ６
(72)発明者 バウス，ステーフェン セー
オランダ国，５６５６ アーアー アインドーフェン，プロフ・ホルストラーン ６

Ｆターム(参考) 5B075 ND03 NK02 NK45 NK49 NR05 PR06 QM02

【要約の続き】

タベース中で相続いているものとし、前記第四の個数の中間文字列に基づいて最終的な個数の結果文字列（３０～３３）を決定し、ここで、前記最終的な個数の結果文字列のそれぞれの文字列は、前記問い合わせ文字列（３４）に比較して高々ある所定の第二の誤り個数の誤りを有するようにするステップを有している。これにより、前記問い合わせ文字列と比較しての完全一致または軽微な誤差のみを含む部分一致、そしてより大きなデータベースにおいても比較的控えめな計算機パワーの使用での高速検索が可能となる。